gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

ianuarie 1999
Vol. 9/1

Lei 10.000

din sumarul numărului:

Derandomizare - algoritmi probabiliști

Problema rutajului - drumuri in sisteme distribuite

Plus Java Serial HTML și altele

T. P.
Aprobat C.N.P.R.
Nr. 103/DC/038/1999
Valabil 1999

Expeditor
s.c. „ComputerPress Agora" SRL
C. P. 230 O.P. 1
RO-4300 Tîrgu-Mureș

INFADRES

Un sfat prinde bine...

...oricând!

Dacă vrei să te orientezi printre miile de produse și tehnologii informatice de ultimă oră

ABONEAZĂ-TE!

Câștigi **30 000** lei și vei primi cadou și CD-urile **WINSTONE '99** și **WINBENCH '99**

COMPUTER PRESS AGORA, Str.Tudor Vladimirescu Nr.63/1, 4300 Tg.Mureș, Tel.065-166516, Fax 065-166290

Vol. 9, Nr. 1 1999

gazeta de informatică

Σ sumar

soluții.. PAG. 4

probleme REZOLVATE
PAGINI REALIZATE DE Clara IONESCU
Algoritmi și implementările lor pentru problemele CEOI'98

probleme.. PAG. 13

propuse pentru REZOLVARE
***
Probleme propuse la IOI'98 Setúbal, Portugalia

interviu.. PAG. 18

Teorie și PRACTICĂ
Interviu cu dl. *Emil Muntean*

focus.. PAG. 22

Alegeri prezidențiale
Mihai SCORȚARU

Derandomizare
Cornelius CROITORU

Problema rutajului
Viorica COVALI

tehno.. PAG. 36

Microprocesoare
Mihai SCORȚARU

Programarea plăcii de sunet
Adrian MONEA

serial.. PAG. 45

Extinderea claselor în Java
Horia GEORGESCU

Pagini WEB cu HTML 3.2
Mihai SCORȚARU

mate.. PAG. 56

Conjectura lui Goldbach
Mihai CIPU
Matematică pură

begin.. PAG. 59

Și piticii concurează
de Sorin GROZA
Programare la cluburile de copii

dialog.. PAG. 63

Chestionar

Poșta redacției

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Editor executiv:
Mircea Sârbu (Târgu-Mureș)
Au mai colaborat:
Emil Muntean (Cluj)
Mihai Scorțaru (Cluj)
Adrian Monea (Cluj)
Cornelius Croitoru (Iași)
Viorica Covali (Chișinău)
Sorin Groza (Aiud)
Mihai Cipu (București)

Tehnoredactare:
Dan Crețu, Călin Borz
Coperta și grafica:
Mircea Drăgoi

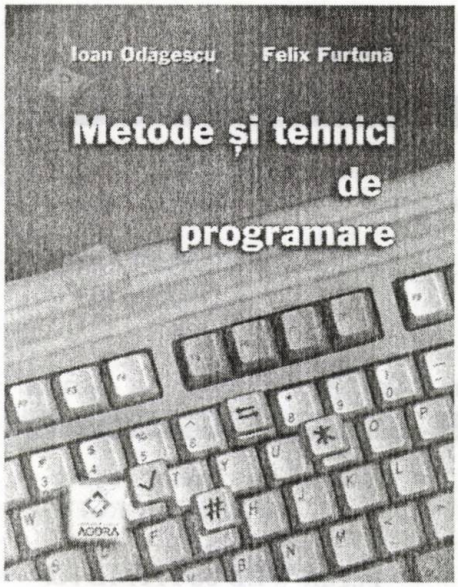
Adresa pentru corespondență:
C.P. 172-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: (065) 166.516
Fax: (065) 166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *S.C. Infopress srl*,
Odorheiu Secuiesc,
tel: 066.216.483
Editor: Computer Press Agora srl
str. T. Vladimirescu nr. 63/1
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director editorial:
Mircea Sârbu
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (*mszabo@agora.ro*)
Denis Mărioara (*mdenis@agora.ro*)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: (01) 330.92.81

Dacă *încă* nu

EȘTI PREGĂTIT PENTRU MÂINE,

trebuie să mergi la ȘCOALĂ, *trebuie* să ÎNVEȚI, îți *trebuie* ... un MANUAL.



Prof. univ. dr. ***IOAN ODĂGESCU***, este absolvent al *Facultăților de Cibernetică, Statistică și Informatică Economică* (***Academia de Studii Economice***) și *Matematică-Mecanică* (***Universitatea București***).

Cu o vechime de peste 25 de ani în învățământul superior de Informatică Economică, membru al Catedrei de Informatică Economică din ASE, a publicat multe lucrări de referință pe diferite domenii, cum ar fi:

În domeniul analizei numerice:
METODE NUMERICE ȘI SUBRUTINE
OPTIMIZAREA ALGORITMILOR

În domeniul informaticii:
VIRUȘI INFORMATICE ȘI BOMBE LOGICE – prima lucrare de acest gen din țară
LIMBAJUL COBOL FĂRĂ PROFESOR
SIMULAREA PROCESELOR ECONOMICE, GRAFICA INTERACTIVĂ PE CALCULATOARE PERSONALE
METODE ȘI TEHNICI DE PROGRAMARE – enunțuri, soluții, probleme propuse <<cartea galbenă>>)

Trebuie remarcat faptul că, domnul profesor a coordonat colectivul care a publicat prima lucrare pe plan național, întitulată OLIMPIADE NAȚIONALE ȘI INTERNAȚIONALE DE INFORMATICĂ

Domnul prof. Ioan Odăgescu este conducător de doctorat în specialitatea Informatică Economică. În cadrul acțiunilor științifice de promovare a tinerelor talente, de aproximativ 20 de ani se ocupă cu organizarea concursurilor de informatică și este autor a multor probleme date la concurs.

Este membru în consiliul ***Societății Române de Informatică*** (SOCROM INFO) și membru ***ACM*** (USA).

* * *

Asist. univ. drd. ***FELIX FURTUNĂ*** este absolvent al *Facultății de Cibernetică, Statistică și Informatică Economică* (***Academia de Studii Economice***, București), specializarea *Informatică Economică*. În prezent este cadru didactic la *Facultatea de Cibernetică Statistică și Informmatică Economică*, catedra de Informatică Economică.

Este coautor al lucrărilor:
METODE ȘI TEHNICI DE PROGRAMARE; OLIMPIADE NAȚIONALE ȘI INTERNAȚIONALE DE INFORMATICĂ
GRAFICA INTERACTIVĂ PE CALCULATOARE PERSONALE

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422] sau la Computer Press AGORA [Tel: 065-166516; 01-3309281].

TREBUIE să se schimbe CEVA?

În ultimul număr al anului se obișnuiește să se facă retrospective. *Ce ne-am propus? Ce am reușit să realizăm?* Dar răspunsul îl știți și voi, cititorii noștri, mai ales că noi, redacția și editura, ne-am împărtășit mereu gândurile.Ne-am propus să relansăm *Gazeta de Informatică*, și am reușit. Ne-am propus s-o "îmbrăcăm" mai frumos și să-i îmbogățim conținutul. Aici vă lăsăm pe voi să apreciați în ce măsură am realizat și acest lucru. Oricum, secțiunile din Ginfo sunt multe și variate: ***Soluții ale problemelor propuse, Probleme propuse, Interviu, Actual, Focus, Tehno, Serial, Internet, Babel, Frontiere, Profil, Mate, Dialog*** și mai nou o rubrică pentru pitici: *Begin.*

Cu riscul că mă veți acuza că vă reproșez ceva, trebuie totuși să menționez că *nu ne-ați asaltat cu articole*. Ne tot întrebăm care ar fi explicația? Să fie revista prea plină de materiale lipsite de interes? Să fie nivelul accesibil doar pentru un număr restrâns de cititori? Să fie de vină faptul că nu se difuzează gratis?

Un lucru e sigur: dacă nu ar exista *Ginfo*, mulți ar simți lipsa ei. Mulți s-ar lamenta: "*Nu avem și noi o revistă...*"

V-am mai scris despre acest subiect: sunt țări unde nu există o revistă a tinerilor "informaticieni", cu toate că multă lume ar dori să fie. De ce nu este? Simplu. Nu e ușor s-o faci.

De ce e utilă?

Să vă povestesc o întâmplare petrecută de câteva zile: o studentă de la medicină mi-a cerut ajutorul pentru că trebuie să finalizeze până la sfârșitul anului o aplicație care să asigure gestionarea tuturor datelor care trebuie prelucrate într-un un cabinet medical: bolnavii cu toate datele personale, cu diagnostic, cu tratamentele efectuate, medicamente, planificări, plăți etc. Pe baza unor cunoștințe acumulate în câteva ore de laborator li se cere să prezinte o aplicație într-un mediu vizual, folosind un editor de texte, un tabelator, un sistem de gestiune a bazelor de date, executabil în acel mediu etc. Iar studenta noastră nu a pus mâna pe o tastatură decât acum câteva săptămâni, la facultate. Nu e corect! - veți exclama cu siguranță. Nu vreau să intru în polemică cu nimeni. Evident, situația este discutabilă, deoarece cei care au inventat această probă cu siguranță au avut un model occidental. Iar în țările respective elevii din liceu se familiarizează cu tehnologia informației an de an, folosind instrumentarul "banal" al informaticii în viața de zi cu zi. Învață să *folosească* editoarele, tabelatoarele, utilitarele cele mai variate, facilitățile multimedia. Pentru ei o asemenea aplicație e *floare la ureche.*

Care este legătura cu Ginfo?

Nu observați? Încercăm să vă ajutăm să ajungeți într-o *stare* în care să nu vă mai speriați de termenii "ciudați" ai tehnologiei informației și să nu vă mai streseze această "cutie neagră", numită calculator. Deschizând cât mai multe ferestre spre cunoștințe care la rândul lor fac parte deja din prezent, familiarizându-vă cu beneficiile aduse de *IT*, intenționăm să vă ajutăm să pășiți cu dreptul în mileniul trei.

Ce așteptăm noi?

De ce nu ne ajutați să ne îmbogățim arsenalul, trimițând unele din articolele apărute în revista școlii? Cu siguranță aveți materiale care ar merita să fie cunoscute de cititorii noștri din întreaga țară. Suntem siguri că organizați sesiuni de comunicări pentru elevi și profesori. De ce nu ne trimiteți câte un rezumat din lucrările cele mai reușite? Sunt o serie de concursuri regionale care ating domenii interesante din informatica aplicată: întreg învățământul duce lipsă de soft educațional, mai ales în limba română, de ce nu am populariza realizările meritorii ale unor elevi? Dacă ceva anume a fost realizat de elevi pentru elevi, cu siguranță răspunde așteptărilor acestora. Copiii care pregătesc materialele pentru concursul "Info-educație", care au activitate în cadrul programului I*AERN și mulți alții sunt așteptați să publice rezultatele lor în paginile revistei.

La mulți ani!

Vă dorim vacanță plăcută și sărbători fericite! Sperăm că nu ne veți uita de tot, oricum, ne "întâlnim" în ianuarie...

Clara Ionescu

SOLUȚIILE problemelor propuse

Ce îi pasionează pe rezolvitorii de probleme din domeniul algoritmicii? În primul rând varietatea enunțurilor. Apoi varietatea rezolvărilor. Vă prezentăm posibile abordări pentru problemele Olimpiadei de Informatică a Europei Centrale și de Est, rezolvări care aparțin autorilor problemelor din Croația. Pentru explicații și implementarea algoritmilor mulțumim colaboratorilor revistei noastre: prof. dr. Horia Georgescu, studentul Mihai Scorțaru și elevul Andrei Vancea.

P089801: Pătrate

Fiecărui al *i*-lea pătrat i se vor asocia două variabile de tip articol P_i și Q_i având câmpurile x și y, reprezentând coordonatele colțurilor din stânga sus, respectiv dreapta jos, ale pătratului. Problema vizibilității pătratelor se reduce la problema vizibilității segmentelor P_iQ_i. Este evident că $Q_i.x=P_i.x+lung_i$, iar $Q_i.y=P_i.y-lung_i$, unde $lung_i$ este lungimea laturii pătratului.

Mai observăm că distanța de la origine la un segment P_iQ_i este $(P_i.x+P_i.y)/\sqrt{2}$. Observația se bazează pe faptul că segmentul P_iQ_i este diagonala unui pătrat, deci întotdeauna formează un unghi de 45 de grade cu abscisa. Rezultă că la compararea distanței a două segmente față de origine este suficient să luăm în calcul cantitățile $P_i.x+P_i.y$.

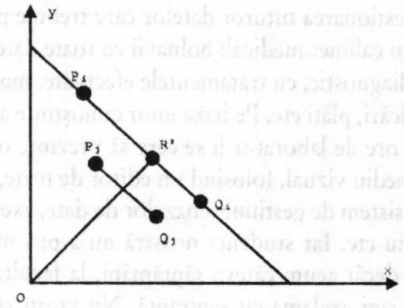
Vom utiliza o funcție Orientare care, apelată pentru două puncte *A* și *B* din primul cadran, va întoarce una dintre valorile 1, 0, -1 după cum panta dreptei *OA* este mai mare, egală sau mai mică decât panta dreptei *OB*.

Începem prin a ordona segmentele P_iQ_i descrescător după pantele lui OP_i.

Considerăm pe rând segmentele P_iQ_i, i=1,...,n și dacă un astfel de segment este vizibil, vom mări cu o unitate variabila n_viz, inițializată cu 0.

Fie P_iQ_i un astfel de segment. Fie R' punctul curent de pe P_iQ_i cu proprietatea că P_iR' este obturată de segmentele P_jQ_j analizate până la acest moment de timp; inițial $R'=P_i$. Deoarece ne interesează numai panta lui OR', vom memora în loc de R' puncte R care sunt actualizate cu anumite puncte Q_j. Atunci pentru segmentul P_iQ_i vom efectua următoarele calcule:

```
R ← Pi
pentru j=1,n execută
  dacă distanța de la O la PiQi este mai mică decât
       cea de la O la PjQj
    atunci {segmentul PiQi nu poate fi obturat de PjQj}
    altfel
      dacă panta lui OR este cuprinsă între panta lui OQj
           și cea a lui OPj, adică dreapta OR taie
           segmentul PjQj (vezi figura)
        atunci R ← Qj

dacă panta lui OQi < panta lui OR
  atunci n_viz ← n_viz+1
```



Ajungem astfel la următorul program:

Listing SQUARES.PAS

```
Program Squares;
const MAX=1000;
type punct=record
            x,y:Longint
          end;

var nr_patrate,n_viz:Integer;
    P,Q:array[1..MAX] of punct;
    f:Text;

procedure Citire;
var i,x,y,lung:Integer;
begin
  Assign(f,'SQUARES.IN'); Reset(f);
  Readln(f,nr_patrate);
  for i:=1 to nr_patrate do
  begin
    Readln(f,x,y,lung);
    P[i].x:=x;
    P[i].y:=y+lung;
    Q[i].x:=x+lung;
    Q[i].y:=y
  end;
  Close(f)
end;

procedure Schimba(a,b:Integer);
var aux:punct;
begin
  aux:=P[a]; P[a]:=P[b]; P[b]:=aux;
  aux:=Q[a]; Q[a]:=Q[b]; Q[b]:=aux
end;

procedure QSort(a,b:Integer);
var i,j,ii,jj,aux:Integer;
begin
  if a>=b then Exit;
  i:=a; j:=b;
  ii:=1; jj:=0;
  while i<j do
  begin
    if P[i].y*P[j].x<P[j].y*P[i].x
    then
    begin
      Schimba(i,j);
      aux:=ii;
      ii:=-jj;
      jj:=-aux
    end;
    i:=i+ii;
    j:=j+jj
  end;
  QSort(a,i-1);
  QSort(i+1,b)
end;
```

```
function Orientare(x1,y1,x2,y2:Longint):Integer;
var res:Longint;
begin
  res:=x1*y2-x2*y1;
  if res>0
  then Orientare:=1
  else
    if res<0 then Orientare:=-1
             else Orientare:=0
end;

procedure Solutie;
var i,j:Integer;
    R:punct;
begin
  QSort(1,nr_patrate);
  n_viz:=0;
  for i:=1 to nr_patrate do
  begin
    R:=P[i];
    for j:=1 to nr_patrate do
    begin
      if P[j].x+P[j].y>=P[i].x+P[i].y
      then
      else
        if Orientare(Q[j].x,Q[j].y,R.x,R.y)<>
           Orientare(P[j].x,P[j].y,R.x,R.y)
        then R:=Q[j]
    end;
    if Orientare(Q[i].x,Q[i].y,R.x,R.y)>0
    then Inc(n_viz)
  end
end;

procedure Scriere;
begin
  Assign(f,'SQUARES.OUT'); Rewrite(f);
  Writeln(f,n_viz);
  Close(f)
end;

Begin
  Citire;
  Solutie;
  Scriere
End.
```

P089802: Cărți de joc

Fie q permutarea de grad n citită de la intrare și p permutarea căutată, adică permutarea inițială care trebuie reconstituită.

Este esențial ca din enunțul problemei să reținem următoarele:

- n este impar (1)
- modul în care a fost construită permutarea p de la care s-a plecat. (2)

De exemplu (2) arată că permutarea inițială p nu este oarecare, ci are proprietatea $p^n=e_n$, unde e_n este permutarea identică de grad n.

Problema cere ca, plecând de la numerele naturale n, s și de la permutarea q, să determinăm acea permutare p de grad n care ridicată la puterea 2^s să dea tocmai q.

Să observăm că (2^s,n)=1 deoarece n este impar. Atunci, conform unui rezultat din cartea de *Algebră de clasa a XII-a*, va exista un număr natural u având proprietatea: $2^s u=1 \pmod n$.

Fie t restul împărțirii lui 2^s la n; el se calculează astfel:

```
t ← 1;
pentru i=1,s execută t ← (t+t) mod n;
```

Acum u poate fi obținut după cum urmează:

```
u ← 1;
cât timp restul împărțirii lui u*t la n diferă de 1
  execută u ← u+1;
```

Rezultă că $2^s u=k*n+1$ pentru un anumit k natural. Deci trebuie determinată permutarea p de grad n care ridicată la puterea $2^s u$ să fie egală cu q^u, adică $p=q^u$.

Suntem conduși astfel la următorul program:

Listing CARDS.PAS

```
Program Cards;
const max=1000;
var n,s:Integer;
    f:Text;
    p,q:array[1..max] of integer;

procedure Citire;
var i:Integer;
begin
  Assign(f,'CARDS.IN'); Reset(f);
  Readln(f,n,s);
  for i:=1 to n do Readln(f,q[i]);
  Close(f)
end;

procedure Solutie;
var i,j:Integer; t,u:Longint;
    temp:array[1..max] of Integer;
begin
  t:=1;
  for i:=1 to s do t:=(t+t) mod n;
  u:=1; while (u*t mod n) <>1 do Inc(u);
  for i:=1 to n do p[i]:=q[i];
  for i:=2 to u do
  begin
    for j:=1 to n do temp[j]:=q[p[j]];
    for j:=1 to n do p[j]:=temp[j]
  end
end;
```

```
procedure Scriere;
var i:Word;
begin
  Assign(f,'CARDS.OUT'); Rewrite(f);
  for i:=1 to n do Writeln(f,p[i]);
  Close(f)
end;

Begin
  Citire;
  Solutie;
  Scriere
End.
```

P089803: Reduceri

Fie a vectorul cu n componente, citit de la intrare și fie t valoarea la care trebuie să ajungem făcând reduceri.

Prin reduceri, unele dintre componentele lui a își schimbă semnul. Căutăm deci vectorul r cu proprietatea:

$r_1a_1+r_2a_2+\ldots+r_na_n=t$ și $r_1,r_2,\ldots,r_n\in\{-1,1\}$ (1)

Fie $a_1+a_2+\ldots+a_n=sum$. Prin adunare cu relația (1) obținem:

$s_1a_1+s_2a_2+\ldots+s_na_n=t+sum$ cu $s_1,s_2,\ldots,s_n\in\{0,2\}$.

Cum $r_1=1$ și $r_2=-1$, rezultă că $s_1=2$, $s_2=0$ și că urmează să determinăm $s_3,\ldots,s_n\in\{0,2\}$ având proprietatea:

$s_3a_3+\ldots+s_na_n=scop$, unde $scop=t+sum-2a_2$

și cu precizarea că s_i va fi 0 dacă și numai dacă r_i este -1.

În continuare vom calcula sumele posibile:

$s_3a_3+\ldots+s_na_n$ și "geneza" lor.

Vom folosi un vector sume indexat de la 0 la 20000 (acum sumele posibile pot fi doar pozitive și egale cu cel mult 20000, deoarece fiecare dintre cei maximum 100 de termeni este cel mult 200). Inițial vom pune sume[0]=-1 și sume[i]=-2 pentru i=1,...,scop. Când o valoare v va fi marcată ca putând fi atinsă, vom pune sume[v]=i, unde $v=v'+a_i$ și v' atinsă.

În acest mod va fi posibilă identificarea acelor elemente a_i care au fost adăugate (de fapt a fost adăugat dublul lor), adică putem determina vectorul r satisfăcând (1).

Mai trebuie să reconstituim din vectorul r pozițiile pe care s-au efectuat reducerile. Pentru aceasta să observăm că dacă este valabilă egalitatea:

$t = a_1 - \ldots - a_j + a_{j-2} + \ldots + a_{i-k} - a_{i-k-1} - \ldots - a_n$

atunci efectuând k reduceri pe poziția i obținem:

$t = a_1 - \ldots - b_i - a_{i-k-1} - \ldots - a_n$

cu $b_i=(\ldots((a_i-a_{i-1})-a_{i-2})-\ldots a_{i-k})$.

Conform observației de mai sus vom pleca cu poziția curentă poz egală cu n și vom repeta succesiv următoarele operații până când ajungem pe prima poziție (mergem mereu de la dreapta la stânga):

- trecem peste toate pozițiile pentru care valoarea r corespunzătoare este -1, micșorând pe poz;
- trecem peste toate pozițiile pentru care valoarea r corespunzătoare este 1, micșorând pe poz; fie nr numărul lor;
- introducem în vectorul reduc de nr ori valoarea curentă a lui poz.

În final se completează cu 1 pozițiile din reduc ce nu au primit valori, corespunzător faptului că acum t se exprimă ca o "sumă" în care doar primul termen are valoarea r corespunzătoare egală cu 1, deci se fac numai reduceri pe prima poziție.

Listing REDUCERI.PAS

```
{$M 65521,0,655360 }
Program Reduceri;
var a,reduc:array[1..100] of Integer;
    n,t:Integer;
    f:Text;

procedure Citire;
var i:Integer;
begin
  Assign(f,'SUBTRACT.IN'); Reset(f);
  Readln(f,n,t);
  for i:=1 to n do Readln(f,a[i]);
  Close(f)
end;

procedure Scriere;
var i:Integer;
begin
  Assign(f,'SUBTRACT.OUT'); Rewrite(f);
  for i:=1 to n-1 do
    Writeln(f,reduc[i]);
  Close(f)
end;

procedure Solutie;
var r:array[1..100] of Integer;
    sume:array[0..20000] of Integer;
    sum,scop:Integer;
    i,j,nr,poz,contor,cont:Integer;
begin
            { Calculul sumelor posibile }
  sum:=0;
  for i:=1 to n do
    Inc(sum,a[i]);
  scop:=t+sum-2*a[1];
  sume[0]:=-1;
  for i:=1 to scop do sume[i]:=-2;
  for i:=3 to n do
  begin
    for j:=scop-1 downto 0 do
      if sume[j]<>-2
      then
      begin
        if sume[j+2*a[i]]=-2
        then sume[j+2*a[i]]:=i;
        if sume[scop]<>-2 then Break
      end;
    if sume[scop]<>-2 then Break
  end;
            { Calculul coeficientilor r }
  r[1]:=1; r[2]:=-1;
  for i:=3 to n do r[i]:=-1;
  nr:=scop;
  repeat
    if nr=0 then Break;
    r[sume[nr]]:=1; nr:=nr-2*a[sume[nr]]
  until false;
          { Calculul pozitiilor pe care }
                  { se fac reducerile }
  poz:=n; contor:=0;
  repeat
    while r[poz]=-1 do Dec(poz);
    if poz=1 then Break;
    cont:=0;
    while r[poz]=1 do
    begin
      Dec(poz);
      Inc(cont)
    end;
    for i:=1 to cont do
    begin
      Inc(contor);
      reduc[contor]:=poz
    end;
    Dec(poz)
  until false;
  for i:=1 to n-1-contor do
  begin
    Inc(contor);
    reduc[contor]:=1
  end
end;

Begin
  Citire;
  Solutie;
  Scriere
End.
```

P089804: Soldați

Etapa 1. Este evident că toate punctele date date (toți soldații) trebuie să "cadă" pe o dreaptă orizontală y=yf.

Drept urmare vom căuta yf cu $\sum_{i=1}^{n} |yf - y_i|$ minim, reprezentând suma distanțelor punctelor la dreapta y=yf.

Tentația de a aborda această problemă folosind cunoștințele de analiză matematică este normală, dar neproductivă în acest caz. Este momentul de a preciza că dacă afirmația "matematica ajută informatica" este cât se poate de corectă, la fel de corectă este afirmația că "*informatica ajută matematica*".

Vom ordona crescător vectorul y, deci acum $y_2 \le y_3 \le \ldots \le y_n$. Pentru aceasta vom utiliza metoda *heapsort* (sortare de ansamble) nu numai pentru a nu intoxica pe cititor

cu sortarea rapidă folosită în programele prezentate, dar și ca prilej pentru a reaminti că metoda heapsort necesită totdeauna un timp de ordinul $O(\log n)$, pe când sortarea rapidă (Quicksort) are aceeași complexitate în timp, dar *numai în medie*.

Să considerăm mai întâi y_1 și y_n. Este clar că suma distanțelor lor la y=yf este minimă dacă drept yf luăm oricare număr întreg din intervalul $[y_1, y_n]$; suma minimă va fi y_n-y_1, oricare ar fi yf astfel ales.

Continuăm cu perechea y_2, y_{n-1}. Rezultă, la fel ca mai sus, că suma distanțelor lor la y=yf este minimă dacă yf este oricare număr întreg din $[y_2, y_{n-1}]$ și că în acest caz suma va fi $y_{n-1}-y_2$.

Se continuă în același mod. Dacă în final ajungem la un singur număr (adică n este impar) este clar că yf va trebui ales ca fiind acel număr. Dacă în final ajungem la două numere (adică n este par), putem lua drept yf oricare număr dintre ele și suma distanțelor la y=yf este diferența lor.

În concluzie yf poate fi ales ca fiind componenta din y de pe poziția (n+1)/2. Să mai observăm că de fapt nu ne interesează efectiv valoarea lui yf, ci numai valoarea minimă a sumei $\sum_{i=1}^{n}|xf-(x_i-i)|$, care se poate calcula după cum urmează conform celor de mai sus:

```
sy ← 0;
p ← 1;
u ← n;
┌■ cât timp p<u
|    sy ← sy + yn - yp;
|    p ← p + 1;
|    u ← u - 1;
└■
```

Etapa 2. Am redus acum problema din plan pe dreaptă și anume pe dreapta y=yf. Punctele de abscise $x_1, x_2, \ldots, x_n$ trebuie deplasate pe pozițiile consecutive (de pe această dreaptă) xf+1, xf+2, ..., xf+n. Rezultă că vom căuta xf cu suma $\sum_{i=1}^{n}|xf+i-x_i|$ minimă,

adică $\sum_{i=1}^{n}|xf-(x_i-i)|$ minimă.

Repetând raționamentul de mai sus referitor la ordonate, vom salva vectorul x în vectorul xinit, vom înlocui fiecare valoare x_i cu x_i-i și vom obține sortare valoarea xf căutată. Acum efortul de a aduce valorile $x_1, x_2, \ldots, x_n$ pe pozițiile xf+1, xf+2, ..., xf+n se poate calcula imediat ca fiind suma valorilor |xf-(xinit$_i$-i)|.

Lăsăm pe seama cititorului să demonstreze că mutarea soldaților poate fi efectuată în sx+sy pași, cu respectarea regulii ca pe fiecare poziție să apară la fiecare moment de timp cel mult un soldat (așa cum cere problema).

Programul este următorul:

Listing SOLDATI.PAS

```
program Soldati;
var a:array[1..10000,1..2] of Integer;
    xinit:array[1..10000] of Integer;
    f:Text;
    n,i,j,p,u,xf,yf:Integer;
    sx,sy:Longint;

procedure sort(poz:Integer);
var i,b:Integer;

  procedure combin(i,n:Integer);
  var j,b,x,y:Integer;
  begin
    j:=i+i;
    x:=a[i,1];
    y:=a[i,2];
    b:=a[i,poz];
    while j<=n do
    begin
      if (j<n) and (a[j,poz]<a[j+1,poz])
      then Inc(j);
      if b>=a[j,poz]
      then Break
      else
      begin
        a[j div 2,1]:=a[j,1];
        a[j div 2,2]:=a[j,2];
        j:=j+j
      end
    end;
    a[j div 2,1]:=x;
    a[j div 2,2]:=y
  end;

begin
                              { creez heap-ul }
  for i:=n div 2 downto 1 do combin(i,n);
                { retrogradez radacina pe   }
                { ultima pozitie curenta i }
  for i:=n downto 2 do
  begin
    b:=a[1,1]; a[1,1]:=a[i,1];
    a[i,1]:=b; b:=a[1,2];
    a[1,2]:=a[i,2]; a[i,2]:=b;
    combin(1,i-1)
  end
end;

Begin
  Assign(f,'SOLDIERS.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do Readln(f,a[i,1],a[i,2]);
  Close(f);
  sort(2);                    { Sortare dupa y }
  sy:=0; p:=1; u:=n;
```

```
  while p<u do
  begin
    Inc(sy,a[u,2]-a[p,2]);
    Inc(p);
    Dec(u)
  end;
  Writeln('sy=',sy:4);
  sort(1);                    { Sortare dupa x }
  for i:=1 to n do
  begin
    xinit[i]:=a[i,1];
    a[i,1]:=a[i,1]-i
  end;
  sort(1);                    { Sortare dupa x }
  xf:=a[(n+1) div 2,1];
  sx:=0;
  for i:=1 to n do
    Inc(sx,abs(xinit[i]-(xf+i)));
  Writeln('sx=',sx:10);
  Writeln('rezultat=',sx+sy:10);
  Assign(f,'SOLDIERS.OUT'); Rewrite(f);
  Writeln(f,sx+sy);
  Close(f)
End.
```

P089805: Șosele

Reformulăm problema în termeni din teoria grafurilor.

Fiind dat un multigraf orientat cu vârfurile 1,...,n, între două vârfuri pot exista mai multe arce (muchii orientate). Fiecărei muchii i se atașează o lungime și o taxă care trebuie plătită la traversarea sa. Se mai dă o sumă de bani k. Se cere cel mai scurt drum (dacă există) de la 1 la N pentru care taxa totală plătită să nu depășească valoarea k.

O muchie este memorată ca o înregistrare (numită muchie) cu câmpurile sursa, tinta, lung, cost cu semnificații evidente.

Numărul muchiilor este nm iar valorile lor sunt memorate în tabloul M. Vom folosi tabloul L(10000) care corespunde unei liste. Fiecare element al listei va fi o înregistrare de tip stare, având câmpurile:

- nod = vârful curent la care s-a ajuns (plecând de la 1);
- cost = taxa care a trebuit plătită pe drumul prin care s-a ajuns la nod;
- dist = lungimea drumului prin care s-a ajuns la nod.

Elementul 0 al listei L conține (∞,∞,∞) pentru ușurarea inserării unui element în listă. Variabila cap păstrează valoarea poziției imediat următoare ultimului element din listă (care de fapt va fi primul scos); deci inițial cap=1.

Vectorul distmin va păstra distanțele minime curente ale drumurilor de la 1 la fiecare vârf, cu respectarea regulii de a nu depăși valoarea k.

Elementele listei L sunt întotdeauna ordonate după cost; deci L(cap-1) va avea cel mai mic cost.

La adăugarea unui triplet (cost, nod, dist) acest triplet nu va fi efectiv inserat dacă există un element din listă cu aceeași valoare pentru nod, dar cu valorile cost și dist mai mari.

După citirea tabloului M de muchii, acestea vor fi ordonate crescător după sursă și pentru fiecare i∈{1,...,n} se identifică poziția start[i] de pe care încep muchiile cu sursa i.

Ideea rezolvării se bazează pe o parcurgere pe lățime (BF) a arborelui, plecând de la lista L, având doar elementul (0,1,0). Se extrage primul element din listă (cel de pe poziția cap-1); fie el (cost, nod, dist). Vârful nu se expandează dacă dist>distmin[nod]. În caz contrar se determină vârfurile j în care se poate ajunge din nod. Se va adăuga la lista L tripletul (cost', j, dist') unde cost' = cost + *costul muchiei* (nod,j) alese și dist' = dist + *lungimea muchiei alese, numai dacă* cost' ≤ k.

Algoritmul se încheie când lista devine vidă. Rezultatul va fi distmin[n].

Listing SOSELE.PAS

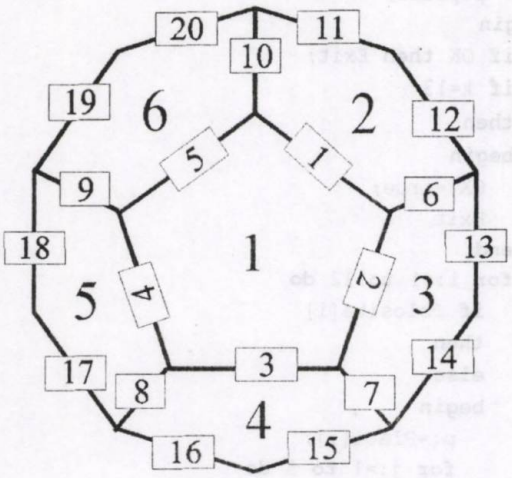
```
{$M 16384,0,655360}
Program Roads;
type stare=record
             cost:Integer;
             nod:Byte;
             dist:Integer
           end;
     muchie=record
              sursa,tinta,lung,taxa:Byte
            end;
       sir=array[1..10000] of muchie;
var cap,K,n,nm:Integer;
    f:Text;
    distmin:array[1..100] of Integer;
    L:array[0..5000] of stare;
    M:^sir;

procedure Citire;
var i,a,b,c,d:Integer;
begin
  New(m);
  Assign(f,'ROADS.IN'); Reset(f);
  Readln(f,K,n,nm);
  for i:=1 to nm do
    with M^[i] do
      Readln(f,sursa,tinta,lung,taxa);
  Close(f)
end;

procedure adaug(s:stare);
var i,j:Integer;
begin
  i:=cap;
  while L[i-1].cost<=s.cost do
  begin
    if L[i-1].nod=s.nod
    then
```

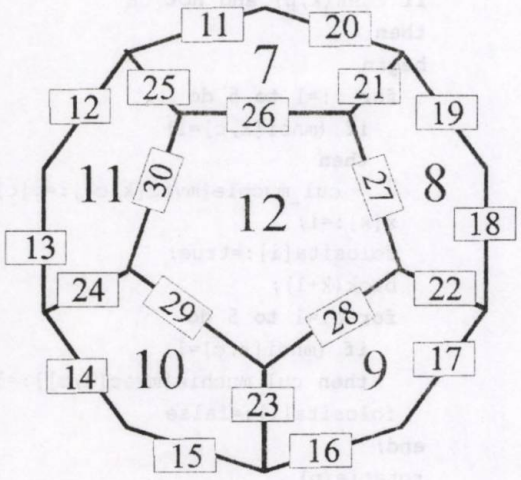
```
    begin
      if L[i-1].dist<=s.dist
      then Exit
      else
        if (L[i-1].cost<s.cost) and
                          (s.dist<L[i-1].dist)
        then
        begin
          L[i-1]:=s; Exit
        end
        else
    end;
    Dec(i)
  end;
  if i=cap
  then
  begin
    L[i]:=s;
    Inc(cap);
    Exit
  end;
  for j:=cap downto i+1 do L[j]:=L[j-1];
  L[i]:=s;
  Inc(cap)
end;

procedure scot(var s:stare);
begin Dec(cap); s:=L[cap] end;

procedure ordonare;
          { Ordonarea muchiilor dupa sursa }
var i:Integer;
    gata:Boolean;
    aux:muchie;
begin
  repeat
    gata:=true;
    for i:=1 to nm-1 do
      if M^[i].sursa>M^[i+1].sursa
      then
      begin
        aux:=M^[i];
        M^[i]:=M^[i+1];
        M^[i+1]:=aux;
        gata:=false
      end
  until gata;
end;

procedure Rezolvare;
var i,j:Integer;
    s,snou:stare;
    start:array[1..100] of Integer;
begin
  distmin[1]:=0;
  for i:=2 to n do distmin[i]:=maxint;
  ordonare;
  j:=1;
  for i:=1 to n do
  begin
    while (M^[j].sursa<i) and (j<=nm) do
      Inc(j);
    if j=nm+1 then Dec(j);
    start[i]:=j
  end;
  L[0].cost:=maxint;
  L[0].nod:=255;
  cap:=1;              { initializare lista }
  s.cost:=0;
  s.nod:=1;
  s.dist:=0;
  adaug(s);
  while cap>1 do
            { cat timp lista este nevida }
  begin
    scot(s);
    if (distmin[s.nod]=0) or
                    (s.dist<distmin[s.nod])
    then
    begin
      distmin[s.nod]:=s.dist;
      i:=start[s.nod];
      while M^[i].sursa=s.nod do
      begin
        if s.cost+M^[i].taxa<=K
        then
        begin
          snou.cost:=s.cost+M^[i].taxa;
          snou.nod:=M^[i].tinta;
          snou.dist:=s.dist+M^[i].lung;
          adaug(snou)
        end;
        Inc(i)
      end
    end
  end
end;

procedure Scriere;
begin
  Assign(f,'ROADS.OUT'); Rewrite(f);
  if distmin[n]<maxint
  then Writeln(f,distmin[n])
  else Writeln(f,'-1');
  Close(f)
end;

Begin
  Citire;
  Rezolvare;
  Scriere
End.
```

P089806: Mingea

Putem considera problema ca o generalizare a problemei colorării hărților, deoarece se urmărește colorarea muchiilor prin intermediul culorilor aflate pe laturile căpăcelelor date. De aceea vom aplica *metoda backtracking*. Am ales varianta recursivă, deși cea nerecursivă era mai bună ca timp și ca facilitate de a opri căutarea când s-a ajuns la o soluție.

Începem prin a numerota muchiile după cum urmează:





Vom memora muchiile vecine fiecărei fețe în vectorul `mvec(12,5)`.

Așezarea căpăcelelor se va face în ordine pe fețele 1, 2, ..., 12. În matricea `mnoi(12,5)` vom memora muchiile care primesc culoare când este așezat un căpăcel pe o față.

Vectorul `x(30)` va păstra culorile atașate muchiilor (dacă o muchie încă nu este colorată, valoarea va fi 3).

Căpăcelele vor fi memorate în vectorul `Placi` având 5 elemente care conțin culorile muchiilor fiecărui căpăcel.

În vectorul `folosita(12)` de tip `Boolean`, păstrăm starea curentă a căpăcelelor (au fost folosite sau nu).

Procedura `Back`, apelată din programul principal prin `Back(1)`, are următoarea formă:

```
procedură Back(k)
  dacă k=13
    atunci scrie_rezultat; stop
    altfel
      pentru i=1,12 execută
      { încerc să pun căpăcelul i pe fața k }
        dacă folosita(i)
          atunci
          altfel p ← Placi(i)
            pentru j=1,5 execută
            { cele 5 poziții ale căpăcelului }
              dacă cont(k,p)
              { condiția de continuare }
                atunci
                { colorez muchiile noi la care ajung }
                  folosita(i) ← true
                  x(k) ← i
                  Back(k+1)
                { "decolorez" muchiile nou colorate }
                  folosita(i) ← false
              { rotește placa p }
```

Funcția de continuare `cont(k,p)` verifică dacă, prin plasarea căpăcelului `p` pe fața `k`, am respectat colorarea muchiilor deja colorate.

Observație:

Modul în care am numerotat muchiile permite reducerea volumului de calcule. Într-adevăr, se observă că vectorii caracteristici din `mnoi` au 1 doar pe ultimele poziții, deci în loc de `mnoi` era suficient să reținem (într-un vector) doar câți de 1 apar (câte muchii vor primi culori la așezarea unui căpăcel pe fața curentă).

Listing MINGEA.PAS

```
program Balls;
const mvec:array[1..12,1..5] of Byte=
        { muchiile adiacente fiecarei fete }
      ( (1,2,3,4,5),(1,10,11,12,6),
        (2,6,13,14,7),(3,7,15,16,8),
        (4,8,17,18,9),(10,5,9,19,20),
        (11,20,21,26,25),(21,19,18,22,27),
        (22,17,16,23,28),(23,15,14,24,29),
        (24,13,12,25,30),(26,27,28,29,30) );
    mnoi:array[1..12,1..5] of 0..1=
{ muchiile noi introduse de fiecare fata }
      ( (1,1,1,1,1),(0,1,1,1,1),(0,0,1,1,1),
        (0,0,1,1,1),(0,0,1,1,1),(0,0,0,1,1),
        (0,0,1,1,1),(0,0,0,1,1),(0,0,0,1,1),
        (0,0,0,1,1),(0,0,0,0,1),(0,0,0,0,0));
```

```
type placa=array[1..5] of 0..2;
var Placi:array[1..12] of placa;
    cul_muchie:array[1..30] of 0..3;
                              { 3 = necolorat }
    x:array[1..12] of Byte;
    folosita:array[1..12] of Boolean;
    OK:Boolean;
    f:Text;
    i:Byte;

procedure Citire;
var i,j:Byte;
begin
  Assign(f,'BALL.IN'); Reset(f);
  for i:=1 to 12 do
  begin
    for j:=1 to 5 do
      Read(f,Placi[i,j]);
    Readln(f)
  end;
  Close(f)
end;

procedure Scriere;
var i:Byte;
begin
  Assign(f,'BALL.OUT'); Rewrite(f);
  if OK
  then
    for i:=1 to 12 do
      Writeln(f,x[i],' ',i)
  else Writeln(f,'-1');
  Close(f)
end;

procedure rotatie(var p:placa);
var i,temp:Byte;
begin
  temp:=p[5];
  for i:=5 downto 2 do
    p[i]:=p[i-1];
  p[1]:=temp
end;

function cont(k:Byte; p:placa):Boolean;
var j,m,color:Byte;
    b:Boolean;
begin
  b:=true;
  j:=1;
  while b and (j<=5) do
  begin
    m:=mvec[k,j];
    color:=cul_muchie[m];
    if (color<>3) and (color<>p[j])
    then b:=false;
    Inc(j)
  end;
  cont:=b
end;

procedure back(k:Byte);
var i,j,c:Byte;
    p:placa;
begin
  if OK then Exit;
  if k=13
  then
  begin
    OK:=true;
    Exit
  end;
  for i:=1 to 12 do
    if folosita[i]
    then
    else
    begin
      p:=Placi[i];
      for j:=1 to 5 do
      begin
        if cont(k,p) and not OK
        then
        begin
          for c:=1 to 5 do
            if (mnoi[k,c]=1)
            then
              cul_muchie[mvec[k,c]]:=p[c];
          x[k]:=i;
          folosita[i]:=true;
          back(k+1);
          for c:=1 to 5 do
            if (mnoi[k,c]=1)
            then cul_muchie[mvec[k,c]]:=3;
          folosita[i]:=false
        end;
        rotatie(p)
      end
    end
end;

Begin
  for i:=1 to 30 do cul_muchie[i]:=3;
  for i:=1 to 12 do folosita[i]:=false;
  Citire;
  OK:=false;
  back(1);
  Scriere
End.
```

PROBLEME propuse pentru REZOLVARE

Iată, am ajuns și la IOI! În Portugalia, la Setubal, băieții din echipa României, împreună cu alți 240 de concurenți s-au întrecut în rezolvarea cât mai performantă a problemelor de mai jos. Încercați și voi să le rezolvați, nu sunt chiar atât de grele...

P099801: Camelot

Cu secole în urmă Regele Arthur și Cavalerii Mesei Rotunde obișnuiau să se întâlnească de Anul Nou. În memoria lor vom considera următorul joc, pentru un singur jucător, în care pe o tablă de șah avem un Rege și câțiva Cavaleri, așezați arbitrar pe pătrățele diferite ale tablei.

Tabla are 8x8 pătrățele:

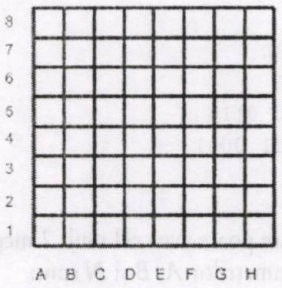


Figura 1. Tabla

Regele poate fi mutat în orice pătrățel vecin din poziția marcată cu • în oricare din cele opt poziții marcate cu O, conform figurii 2, astfel încât să rămână pe tablă.

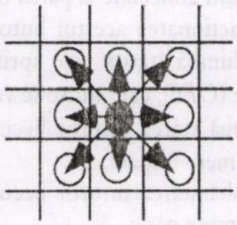


Figura 2. Mutările posibile ale Regelui

Un Cavaler poate fi mutat din poziția marcată cu • în oricare din cele opt poziții marcate cu O, conform figurii 3 astfel încât să rămână pe tablă.

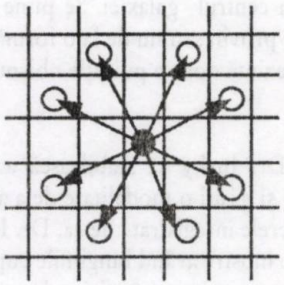


Figura 3. Mutările posibile ale unui Cavaler

Pe parcursul jocului, un pătrățel poate fi ocupat de mai multe piese, în același timp. Pătrățelele sunt suficient de mari încât permit așezarea pe ele a oricâte piese.

Scopul jucătorului este de a muta toate piesele pe un același pătrățel cu numărul minim posibil de mutări, conform regulilor descrise. Dacă Regele și unul sau mai mulți Cavaleri se află pe același pătrățel, jucătorul are voie să mute Regele împreună cu unul dintre Cavaleri. Ei se vor muta împreună, de aici încolo, ca și cum ar fi un singur Cavaler, până la pătrățelul final. Mutarea unui Cavaler împreună cu Regele se contorizează ca fiind una singură.

Cerințe

Scrieți un program care determină numărul minim de mutări necesare pentru a muta toate piesele pe un singur pătrățel.

Date de intrare

Fișierul de intrare `CAMELOT.IN` conține configurația inițială a tablei sub forma unui șir de caractere. Stringul conține o secvență de cel mult 64 elemente distincte, primul element (format din două caractere) precizând poziția Regelui, restul fiind descrieri ale pozițiilor Cavalerilor. Fiecare element corespunzător unei poziții este format dintr-o literă urmată de o cifră. Litera precizează coloana, iar cifra indică linia.

Exemplu:

```
D4A3A8H1H8
```

În acest exemplu Regele se află pe poziția D4 și avem patru cavaleri pe pozițiile A3, A8, H1 și H8.

Date de ieșire

Fișierul de ieșire **CAMELOT.OUT** va conține pe o singură linie numărul minim de mutări prin care jucătorul poate să adune toate piesele pe un același pătrățel.

Corespunzător exemplului dat, conținutul fișierului de ieșire trebuie să fie:

```
10
```

Restricții

0 ≤ numărul cavalerilor ≤ 63

P099802: Contact

Dr. Astro Insky lucrează într-un centru de radiotelescopie. Recent, ea a înregistrat câteva semnale pulsatoare ciudate sosite din centrul galaxiei. Se pune problema dacă aceste semnale provin sau nu de la o formă de viață inteligentă sau reprezintă doar o pulsație obișnuită a stelelor.

Cerințe

Ajutați-o pe Dr. Insky să stabilească adevărul privind aceste semnale și găsiți o modalitate de a analiza mostrele de biți din fișierele înregistrate de ea. Dr. Insky își propune să depisteze mostre având lungimile cuprinse între *A* și *B* inclusiv, care se pot repeta în fișierul înregistrat de ea. Se vor căuta cele mai mari *N* frecvențe (număr de apariții) distincte. Mostrele pot fi și suprapuse. Frecvențele sunt numere mai mari decât zero.

Date de intrare

Conținutul și formatul fișierului **CONTACT.IN** este următorul:

Prima linie - conține un număr întreg *A*, reprezentând lungimea minimă a mostrei;

A doua linie - conține un număr întreg *B*, reprezentând lungimea maximă a mostrei;

A treia linie - conține un număr întreg *N*, reprezentând numărul de frecvențe distincte care se vor căuta;

A patra linie - conține o secvență formată din caractere 0 și 1, terminată prin caracterul 2.

Exemplu:

```
2
4
10
0101001001000100011110110000101001100111100001001001111001000000002
```

În acest exemplu se cer cel mult primele 10 cele mai mari frecvențe de mostre, având lungimile cuprinse între 2 și 4 inclusiv, existente în secvența de biți.

Cea de a patra linie din fișierul de intrare este afișată pe mai multe linii pentru a putea fi listată în enunț. În acest exemplu, mostra 100 apare de 12 ori, iar mostra 1000 apare de 5 ori. Frecvența maximă de apariție o are mostra 00.

Date de ieșire

Fișierul de ieșire **CONTACT.OUT** va avea cel mult *N* linii, fiecare conținând cele mai mari *N* frecvențe și mostrele corespunzătoare. Afișarea se va face în ordinea descrescătoare a frecvențelor. Fiecare linie va avea următorul format:

frecvență mostră mostră ... mostră

unde *frecvență* reprezintă numărul de apariții a mostrelor care sunt scrise pe linia respectivă. Mostrele de pe o aceeași linie vor fi scrise în ordinea descrescătoare a lungimii, iar în cazul unor lungimi egale, în ordine numerică descrescătoare. În cazul în care există mai puțin decât *N* frecvențe diferite, fișierul de ieșire va conține mai puțin de *N* linii.

Exemplu:

Pentru fișierul de intrare din exemplu, fișierul de ieșire va conține:

```
23 00
15 10 01
12 100
11 001 000 11
10 010
8 0100
7 1001 0010
6 0000 111
5 1000 110 011
4 1100 0011 0001
```

Restricții

Fișierul de intrare poate avea cel mult 2 megabytes.

Valorile parametrilor *A*, *B* și *N* sunt:

$0 < N \leq 20$

$0 < A \leq B \leq 12$

P099803: Orgă de lumini

La masa festivă a IOI'98 avem *N* becuri numerotate de la 1 la *N*. Becurile sunt conectate la patru butoane:

butonul 1 - prin acționarea acestui buton, toate becurile își schimbă starea: cele aprinse (*ON*) se vor stinge (*OFF*), cele stinse se vor aprinde;

butonul 2 - schimbă starea tuturor becurilor numerotate cu numere impare;

butonul 3 - schimbă starea tuturor becurilor numerotate cu numere pare;

butonul 4 - schimbă starea becurilor numerotate cu numere de forma $3K+1$ ($K \geq 0$), adică: 1, 4, 7, ...;

Contorul *C* înregistrează numărul total de acționări ale butoanelor. La începutul petrecerii toate becurile sunt aprinse (*ON*) și contorul *C* este 0.

Cerințe

Aveți la dispoziție o valoare dată pentru contorul *C* și informații despre starea finală a câtorva becuri. Scrieți un program care determină toate configurațiile posibile distincte a celor *N* becuri ce se pot obține din datele inițiale.

Date de intrare

Fișierul `PARTY.IN` are patru linii și conține numărul de becuri *N*, numărul total de acționări *C* ale butoanelor și starea unor becuri în configurația finală.

Prima linie conține numărul *N*, linia a doua valoarea finală a contorului *C*, cea de a treia linie conține lista numerelor de ordine ale becurilor care trebuie să fie aprinse în configurația finală (separate printr-un singur spațiu). Linia se termină cu numărul întreg -1. Cea de a patra linie conține lista numerelor de ordine ale becurilor care trebuie să fie stinse în configurația finală. Această linie are același format ca linia a treia.

Exemplu:

```
10
1
-1
7 -1
```

În acest exemplu avem 10 becuri și doar o singură acționare de buton. Becul numerotat cu 7 trebuie să fie stins în configurația finală.

Date de ieșire

Fișierul de ieșire `PARTY.OUT` trebuie să conțină toate configurațiile finale posibile ale stărilor becurilor. Fiecare linie conține o singură configurație. O configurație se va scrie în fișier o singură dată. Ordinea de scriere a configurațiilor este arbitrară.

Fiecare linie conține *N* caractere. Primul caracter reprezintă starea becului numerotat cu 1, iar ultimul caracter, starea becului numerotat cu *N*. Valoarea 0 (zero) corespunde stării *OFF* (stins), iar valoarea 1 (unu) corespunde stării *ON* (aprins).

Exemplu:

```
0000000000
0110110110
0101010101
```

În acest exemplu există trei configurații finale posibile:
- fie toate becurile sunt stinse (*OFF*);
- fie becurile numerotate cu 1, 4, 7, 10 sunt stinse (*OFF*) și becurile 2, 3, 5, 6, 8, 9 sunt aprinse (*ON*);
- fie becurile numerotate cu 1, 3, 5, 7, 9 sunt stinse (*OFF*) și becurile 2, 4, 6, 8, 10 sunt aprinse (*ON*).

Restricții

Valorile parametrilor *N* și *C* sunt:
$10 \le N \le 100$, $1 \le C \le 10000$

Numărul becurilor impuse a fi aprinse atât în configurația inițială cât și în cea finală este mai mic sau egal cu 2.

Pentru orice fișier de intrare există cel puțin o configurație finală posibilă.

P099804: Dreptunghiuri

Pe un perete se lipesc mai multe postere, fotografii și alte desene de forme dreptunghiulare. Laturile sunt paralele cu axele de coordonate. Dreptunghiurile se pot suprapune parțial sau integral. Lungimea marginilor (contururilor) reuniunii tuturor dreptunghiurilor se va numi *perimetru*.

Cerințe

Să se calculeze perimetrul.

În figura 1 avem un exemplu cu 7 dreptunghiuri:

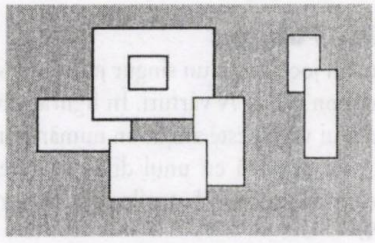


Figura 1. O mulțime de 7 dreptunghiuri

Perimetrul căutat este suma lungimilor din figura 2:

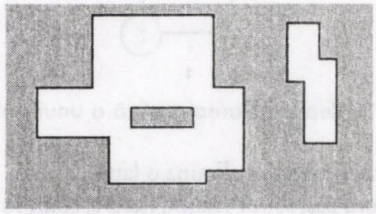


Figura 2. Perimetrul reuniunii dreptunghiurilor

Date de intrare

Pe prima linie a fișierului de intrare `PICTURE.IN` se află un număr ce reprezintă numărul figurilor dreptunghiulare. Pe următoarele linii sunt scrise coordonatele întregi ale colțurilor stânga-jos și dreapta-sus ale dreptunghiurilor. Valorile acestor coordonate sunt date sub forma unor perechi ordonate (prima coordonată este x, iar a doua este y).

Exemplu:

```
7
-15 0 5 10
-5 8 20 25
15 -4 24 14
0 -6 16 4
2 15 10 22
30 10 36 20
34 0 40 16
```

Acest fișier corespunde exemplului din figura 1.

Date de ieșire

Fișierul de ieșire `PICTURE.OUT` trebuie să conțină o singură linie pe care este scris un număr întreg nenegativ reprezentând perimetrul obținut pentru dreptunghiurile din fișierul de intrare.

Corespunzător exemplului dat, în fișierul de ieșire trebuie scris:

```
228
```

Restricții

$0 \leq$ numărul de dreptunghiuri < 5000

Toate valorile coordonatelor sunt numere întregi și aparțin intervalului [-10000, 10000] și fiecare dreptunghi are arie pozitivă. Valoarea numerică a rezultatului necesită o reprezentare pe 32 de biți.

P099805: Poligon

Poligon este un joc pentru un singur jucător și se realizează pe un poligon având N vârfuri. În figura 1 avem $N = 4$ vârfuri. Fiecărui vârf îi este atașat un număr întreg, iar fiecare latură este marcată cu unul din caracterele + (adunare), respectiv * (produs). Laturile sunt numerotate de la 1 până la N.

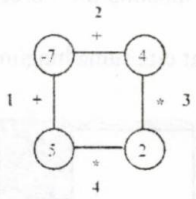


Figura 1. Reprezentarea grafică a unui poligon

La *primul pas* se va elimina o latură.

Următorii pași se vor realiza, fiecare, conform descrierii:

- se alege o latură E și cele două vârfuri V_1 și V_2 legate prin latura E;
- acestea se înlocuiesc cu un vârf nou căruia i se atașează rezultatul operației corespunzătoare semnului din dreptul laturii E aplicate celor două valori atașate vârfurilor V_1 și V_2.

Jocul se termină când nu mai sunt laturi de eliminat și a rămas un singur vârf căruia îi este atașat *scorul* jocului.

Exemplu:

Fie poligonul din figura 1. Dacă jucătorul la *primul pas* șterge latura 3, efectul este vizibil în figura 2:

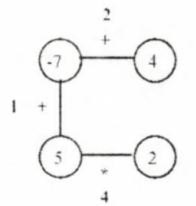


Figura 2. Eliminarea laturii 3

La următorul pas, jucătorul alegând latura 1, va obține:

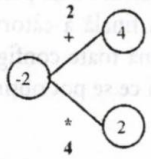


Figura 3. Efectul pasului doi

apoi, alegând latura 4, va obține:

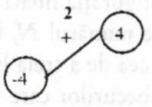


Figura 4. Efectul pasului trei

și în final, alegând latura 2 va obține scorul final 0.

Figura 5. Efectul ultimului pas

Cerințe

Să se determine cel mai mare scor posibil de obținut pentru un poligon dat și să se formeze lista acelor laturi care, dacă sunt eliminate la primul pas, conduc la variante de realizări ale jocului, cu același scor final.

Date de intrare

Fișierul de intrare `POLYGON.IN` descrie un poligon având N vârfuri. Fișierul are două linii. Pe prima linie este scris numărul N. A doua linie conține marcajele atașate laturilor și valorile atașate vârfurilor în felul următor: primul element este marcajul atașat laturii 1, urmează valoarea numerică atașată vârfului dintre prima și a doua latură, apoi avem marcajul celei de-a doua laturi și așa mai departe, ultimul element reprezentând valoarea numerică atașată vârfului dintre laturile având numerele de ordine N și 1. Aceste elemente sunt separate prin câte un spațiu. Marcajul atașat unei laturi poate fi litera `t` (reprezentând +) sau litera `x` (reprezentând *).

Exemplu:

```
4
t -7 t 4 x 2 x 5
```

Aceste date corespund poligonului din figura 1.

Date de ieșire

Pe prima linie a fișierului `POLYGON.OUT` se va scrie scorul maxim posibil, obținut pentru poligonul dat. Pe cea de-a doua linie se va scrie lista numerelor de ordine ale laturilor care, dacă sunt eliminate la primul pas, conduc la variante de realizări ale jocului cu același scor final, scris pe prima linie. Numerele de ordine ale laturilor se vor scrie în ordine crescătoare, separate printr-un singur spațiu.

Corespunzător exemplului dat, conținutul fișierului de ieșire trebuie să fie:

```
33
1 2
```

Restricții

$3 \leq N \leq 50$

Pentru orice secvență de pași, numerele atașate vârfurilor aparțin intervalului [-32768, 32767].

P099806: Cer înstelat

Noaptea, pe cer, stelele apar în grupări având diferite forme. O *grupare* este o mulțime nevidă de stele învecinate, adiacente pe orizontală, verticală sau diagonală. O grupare nu poate să aparțină unei alte grupări mai mari.

Două grupări se consideră *similare* dacă ele au aceeași formă și același număr de stele, indiferent de poziția și orientarea grupărilor. Numărul posibil de orientări pentru o grupare este opt, conform figurii 1.

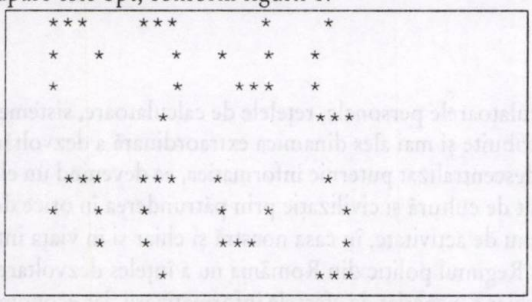


Figura 1. Cele opt grupări similare

Cerului înstelat îi corespunde o *hartă* reprezentată printr-o matrice bidimensională, având elemente 0 și 1. Valoarea 1 semnifică prezența unei stele. Dacă pe hartă într-o poziție nu este stea, valoarea corespunzătoare în matrice este 0.

Cerințe

Dându-se o hartă a cerului, marcați cu litere mici toate grupările determinate. Grupările similare trebuie să fie marcate cu aceeași literă; cele nesimilare se vor marca cu litere diferite.

Veți marca o grupare înlocuind valorile 1 cu litere mici, folosind aceeași literă mică pentru toată gruparea.

Date de intrare

Pe primele două linii ale fișierului **STARRY.IN** sunt scrise două numere, reprezentând numărul de coloane *W*, respectiv numărul de linii *H* ale hărții.

Harta este dată pe următoarele *H* linii, a câte *W* caractere pe fiecare linie.

Exemplu:

```
23
9
00000000110000001000000
01110010000100101000010
01010000000000101101000
01110010101110001001011
00000110001010011101010
00000000001110000001000
01111101000000010001000
00000000000111110100010
00111000000001010110000
```

În acest exemplu harta are 23 coloane și 9 linii. Pentru a fi exemplul mai clar, vă precizăm că acestui fișier de intrare îi corespunde următoarea imagine a cerului.

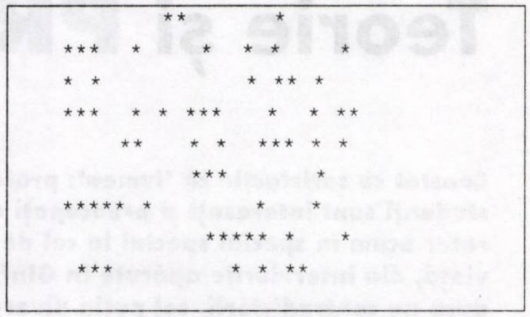


Figura 2. Imaginea cerului

Date de ieșire

Fișierul de ieșire **STARRY.OUT** conține aceeași hartă ca fișierul de intrare **STARRY.IN**, cu deosebirea că grupările sunt marcate conform cerințelor.

Exemplu:

```
00000000ff000000c000000
0aaa00d0000d00f0c0000d0
0a0a0000000000f0cc0b000
0aaa00g0d0aaa000c00b0gg
00000gg000a0a00ccc0b0g0
0000000000aaa000000b000
0bbbbb0d0000000c000b000
0000000000ccccc0g000d0
00eee00000000c0c0gg0000
```

Acest conținut reprezintă unul din rezultatele posibile ale fișierului de ieșire; altele se pot obține marcând grupările cu alte litere.

Imaginea următoare corespunde fișierului de ieșire prezentat mai sus.

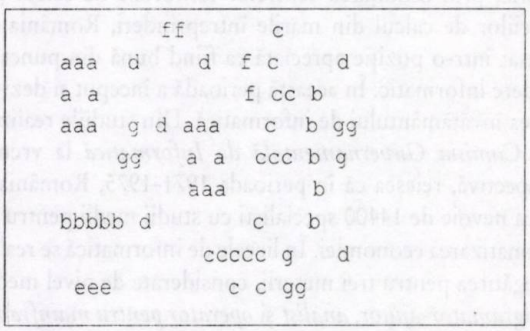


Figura 3. Imagine cu grupările marcate

Restricții

$0 \leq W$ (numărul de coloane ale matricei) ≤ 100

$0 \leq H$ (numărul de linii ale matricei) ≤ 100

$0 \leq$ Numărul grupărilor ≤ 500

$0 \leq$ Numărul grupărilor nesimilare ≤ 26 (a..z)

$1 \leq$ Numărul stelelor dintr-o grupare ≤ 160

În dialog cu domnul profesor dr. Emil Muntean

Teorie și PRACTICĂ

Clara Ionescu

Constat cu satisfacție că "lumea": profesori universitari și de liceu, elevi și studenți sunt interesați și preocupați de viitorul învățământului românesc, mă refer acum în special special la cel de informatică. Și așa cum se întâmplă în viață, din interviurile apărute în GInfo dar și în PC Report se desprind păreri, dacă nu contradictorii, cel puțin divergente.

Înainte de a vă "provoca" în sensul unei confruntări concrete cu părerile exprimate în interviuri anterioare, vă pun o întrebare generală: Care este părerea Dvs. despre învățământul de informatică din România?

În general am o părere bună. Învățământul de informatică din România are câteva trăsături specifice, puternic influențate de situația politică, economică și socială a țării, caracterizată prin perioade total diferite și contradictorii. În analizele făcute de Dvs., prin interviurile din *GInfo*, foarte interesante și la obiect, v-ați preocupat mai mult de perioada anilor '90. Dar concepțiile, mentalitatea, strategia și obiceiurile multora dintre profesorii care funcționează azi în învățământul de informatică, s-au format în anii '70 și '80. Ori acestea au fost două perioade diferite de dezvoltare a informaticii românești și diferite mult de anii '90.

În anii '70, prin asimilarea tehnologiei de fabricare a calculatoarelor și fabricarea mainframe-urilor din seria FELIX și a minicalculatoarelor din seria Independent și Coral, prin înființarea centrelor teritoriale de calcul și a oficiilor de calcul din marile întreprinderi, România s-a plasat într-o poziție apreciată ca fiind bună din punct de vedere informatic. În această perioadă a început și dezvoltarea învățământului de informatică. Din studiile realizate de *Comisia Guvernamentală de Informatică* la vremea respectivă, reieșea că în perioada 1971-1975, România ar avea nevoie de 14400 specialiști cu studii medii pentru informatizarea economiei. În liceele de informatică se realiza pregătirea pentru trei meserii, considerate de nivel mediu: *programator-ajutor*, *analist* și *operator pentru mainframe-uri și minicalculatoare*. Aceasta a fost una din trăsăturile esențiale a învățământului liceal românesc de informatică și așa a rămas mulți ani, până în prezent. Dar aș vrea să menționez că absolvenții au fost bine pregătiți.

În anii '80 informatica s-a schimbat. Microprocesorul, calculatoarele personale, rețelele de calculatoare, sistemele distribuite și mai ales dinamica extraordinară a dezvoltării au descentralizat puternic informatica, ea devenind un element de cultură și civilizație prin pătrunderea în orice domeniu de activitate, în casa noastră și chiar și în viața intimă. Regimul politic din România nu a înțeles dezvoltarea, ba chiar s-a speriat de efectele informatizării, iar economia românească nu a putut ține pasul cu dinamica dezvoltării informaticii. Acest fenomen s-a accentuat în anii '90, când România apare ca o țară slab informatizată, cu o dotare mult sub nivelul mediu mondial, în schimb cu un număr destul de mare de specialiști bine pregătiți și bine plătiți pe piața mondială a forței de muncă din domeniu. Ultimele date din mass-media arată că anual de pe băncile facultăților românești ies 5000 specialiști față de cei 24000 ai Statelor Unite. Față de India, al doilea exportator mondial de software, România are de 7 ori mai mulți specialiști la mia de locuitori, iar exportul nostru de software este de numai câteva zeci de milioane USD. Încă o anomalie românească.

S-a spus de câteva ori că datorită apariției liceelor de informatică în România acum 27 de ani, și datorită funcționării lor pe parcursul anilor la standarde care au avut ca efect răspândirea lor, s-a generat un conflict greu de rezolvat între învățământul de informatică liceal și cel universitar. Ați putea să ne vorbiți despre acest "conflict"?

Nu cred că trebuie să vorbim despre un "conflict", ci mai degrabă de niște necorelări create de anumite situații speciale (și necorectate la timp). Am spus că România a avut o perioadă specială de dezvoltare a informaticii în anii '70. În această perioadă s-au înființat liceele de informatică (în foarte multe țări nici nu există un învățământ preuniversitar de informatică) și, fiind sub tutela Comisiei Guvernamentale de Informatică au beneficiat de o dotare bună. În comparație cu învățământul universitar au pus un

pede locul în care se pregăteau buni programatori, unde se realiza o bună practică de programare și un climat corespunzător pentru munca în echipă a programatorului.

Oare chiar are dreptate doamna profesoară Irina Athanasiu când spune că "Problema este că nu studenții români câștigă premii; cei care câștigă sunt câțiva studenți și elevi din România. Performanța olimpicilor este rezultatul efortului personal, a faptului că - pe baza unor calități native deosebite, cu un efort susținut - se poate ajunge la performanță. "?

Eu interpretez aprecierile doamnei profesoare univ. dr. ing. *Irina Athanasiu* în interviul acordat d-lui *Mircea Sârbu* și publicat în *GInfo vol. 8/4-5, 1998* în sensul că nivelul învățământului liceal, și nu numai, trebuie apreciat în primul rând prin rezultatele absolvenților "obișnuiți", prin capacitatea acestora de a rezolva probleme (nu de a reproduce cunoștințe din cărți) multe cu un pronunțat caracter practic și nu prin rezultatele olimpicilor. Aceasta înseamnă însă că pentru formarea olimpicilor nu se munceşte în licee și universități. Cunoașteți și Dvs, mai bine decât mine, pentru că ați fost implicată direct în organizarea pregătirii loturilor olimpice, cât este de laborioasă și de complexă o asemenea pregătire. Mi se pare normal să existe o asemenea activitate pentru că fără elite nu se poate face nimic, ori performanța cere calități native plus multe cunoștințe și antrenamente, oferite de școală și centre de pregătire.

În calitatea Dvs. de profesor universitar la Universitatea "Babeș-Bolyai" din Cluj, sunteți preocupat desigur, de pregătirea teoretică și practică a studenților, care după absolvire, în majoritate vor căuta un loc de muncă mai întâi la o firmă. Abia în cazul unei "căutări fără succes" se vor îndrepta spre învățământ. (Au dreptate cei care sunt nemulțumiți de aprecierea socială a meseriei de dascăl!) Ce ne puteți spune despre pregătirea studenților în acest sens? Vor fi ei pregătiți teoretic și practic pentru meseria de profesor?

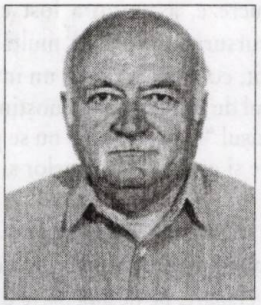
Da, cred că pot fi și sunt pregătiți pentru meseria de profesor de informatică. Sunt necesare cel puțin două componente pentru a deveni un profesor bun. În primul rând vocația, pasiunea de a forma caractere, iar în al doilea rând cunoștințele și practica necesare în informatică. Eu cred că profesorul de informatică mai trebuie în plus să înțeleagă caracterul extraordinar de dinamic al acestei profesii și să fie educat și convins că în domeniul informaticii se poate găsi totdeauna un job în plus care poate rotunji în bună măsură veniturile extrem de modeste și slaba apreciere socială a meseriei de dascăl. În foarte multe țări profesorul de informatică are o mică firmă formată din el și eventual unii membri ai familiei (în special copii-elevi) prin care își valorifică munca proprie de programare, de tehnoredactat, de vânzări de programe sau calculatoare, de consultanță etc.

Este absolut necesar ca învățământul nostru universitar să formeze o educație în această direcție. Va fi în beneficiul elevilor. Ei vor avea în față un profesor mai puțin lipsit de anumite griji materiale și mai cunoscător al cerințelor societății față de informatică.

În multe țări din Europa viitorii profesori realizează "practică pedagogică" pe durata mai multor ani de studiu și această "disciplină" este una dintre cele mai importante. Din păcate, la noi așa se pare că practica viitorilor profesori nu este apreciată ca fiind ceva important, iar profesorul de metodică nu se bucură de același prestigiu ca ceilalți profesori de la alte catedre. Nu există posibilitatea de a verifica dacă profesorul care primește studenți în practică pedagogică este (sau nu) capabil de a-și îndeplini sarcinile la nivelul dorit în sensul de model. Chiar dacă se apelează pentru informații la Inspectoratele Școlare, nu se pot respecta întotdeauna aceste cerințe, din lipsă de cadre. Și uite așa ajungem în celebrul "cerc vicios".

Sunteți sub influența noii Dvs. activități, aceea de profesor de metodica informaticii la Facultatea de Matematică

Carte de vizită



Domnul profesor univ. dr. Emil Muntean este cadru didactic al Facultății de Matematică și Informatică a Universității "Babeș-Bolyai" din Cluj, unde a predat și predă cursuri de limbaje de programare, arhitecturi de calculatoare, sisteme expert, birotică și rețele de calculatoare.

Domnul prof. univ. dr. Emil Muntean a funcționat ca cercetător și director al Filialei din Cluj a Institutului pentru Tehnică de Calcul de la înființarea acesteia, în 1968 și până în anul 1990. În anul 1971, Emil Muntean și-a adus contribuția la înființarea primelor licee de informatică, a fost profesor "asociat" al Liceului de Informatică din Cluj și autor al primului manual de informatică pentru licee - *Inițiere în prelucrarea automată a datelor*, Cluj, 1972. A participat la numeroase proiecte de cercetare și a publicat numeroase publicații și cărți de specialitate atât în țară cât și în străinătate. A înființat editurile Microinformatica și Promedia și a editat un număr mare de manuale și cărți de informatică.

accent mai mare pe dezvoltarea unor deprinderi practice. Ajungând repede să fie cotate ca licee "foarte bune", liceele de informatică au atras elevi foarte buni și ca urmare, după absolvire, majoritatea licențiaților nu s-au angajat să lucreze, așa cum a fost elaborată strategia, ci au urmat cursuri universitare, mulți în facultăți de profil. Aici au găsit, cum era și firesc, un învățământ care pleacă de la nivelul de *începător* al cunoștințelor de informatică. De aici așa zisul "conflict" care nu se poate evita fără o bună înțelegere și corelare a formelor și treptelor învățământului.

Analiza făcută de Dvs. punctează anumite situații care pricinuiesc probleme pe de o parte cadrelor didactice universitare, pe de altă parte unor studenți care vin la secția informatică a facultății, fără ca în liceu să fi urmat studiile într-o clasă de informatică. Credeți că s-a greșit cândva, undeva?

Da și nu. *Nu,* pentru că acum 27 de ani era foarte dificil să se prevadă ritmul de dezvoltare a informaticii. *Da,* pentru că nici în prezent nu se ține cont de impactul extraordinar al informaticii în viața economică și socială (mă refer în primul rând la putere, la clasa politică, și într-o anumită măsură chiar și la comunitatea academică). În definitiv cine este vinovat pentru faptul că profilul de informatică este recunoscut abia în 1998, și atunci numai pe jumătate, pentru că în învățământul universitar specializarea prin doctorat în informatică nu este încă recunoscută, iar hotărârea de Guvern (nr.58/02.1998) privind "*Aprobarea strategiei naționale de informatizare și implementare în ritm accelerat a societății informaționale*", deși are opt luni de la aprobare încă nu se aplică? Probabil, sunt puțini cei care au auzit de celebra "lege" a lui *G. Moore: la fiecare 18 luni numărul tranzistorilor integrați într-un microprocesor se dublează*, previziune care se respectă de mai bine de 30 de ani și probabil va fi valabilă încă 10-15 ani. Eu cred că pe de o parte informaticienii nu sunt lăsați să-și reformeze învățământul și statutul, iar pe de altă parte e posibil să le lipsească și lor curajul și puterea necesară.

Nu cred că este cazul să criticăm pe cei care au luat decizia înființării acestor licee acum 27 de ani. Atunci, scopul era de a pregăti cadre mediu specializate de care economia avea nevoie. Dar printr-o "minune" aceste școli au atras cei mai talentați absolvenți ai școlilor generale, asigurând liceelor de informatică promovabilitate 100%, succes școlar în foarte multe sensuri (număr mare de admiși în învățământul superior, rezultate de excepție la diverse concursuri școlare etc.). Aceste aspecte atractive au ridicat "cota" școlilor respective, a absolvenților lor, în consecință au apărut foarte multe solicitări de a înființa noi licee (clase) de informatică. Ce credeți, se justifică în continuare existența lor? Reamintesc că în planul de învățământ aceste clase au 8 ore de informatică pe săptămână (25% din totalul de ore) și în programa școlară au algoritmică, programare, informatică economică, informatică aplicată (desktop publishing), rețele.

Existența liceelor de informatică poate fi în continuare justificată pentru că au o foarte bună tradiție, rezultate bune și un nucleu de profesori foarte buni, care ar putea contribui la formarea profesorilor pentru disciplina ***Tehnologia informației***, de care va fi mare nevoie odată cu introducerea ei în toate profilurile învățământului liceal. Profilul actualelor licee de informatică va trebui corelat și mereu actualizat cu nevoile pieței și cu învățământul universitar de informatică. Apariția colegiilor de tehnologia informației și perspectiva unui învățământ universitar în trepte, impun necesitatea regândirii învățământului liceal de informatică din România.

Acum 25 de ani ați fost profesor "asociat" la Liceul de Informatică din Cluj. Ce amintiri păstrați despre anii în care (în timpul Dvs. liber) ați predat la liceu?

Îmi aduc aminte cu nostalgie de perioada înființării liceelor de informatică, de lupta dură dată de reprezentanții Comisiei Guvernamentale de Informatică cu inspectorii Ministerului Învățământului pentru înființarea acestor licee. Nu pot vorbi despre acest subiect, fără să menționez rolul uriaș jucat de General-maior (retr.) ing. ***Mihai Varia***, *cavaler al ordinului Mihai Viteazul*, care a coordonat programul comisiei pentru înființarea învățământului liceal de informatică și care, în vara anului 1971, numai pe baza unor protocoale de colaborare semnate cu rectorii universităților din Cluj, Iași și Timișoara a organizat admiterea și a deschis primele licee de informatică din România. Despre acele întâmplări, devenite acum "istorie", cititorul acestor rânduri poate răsfoi volumul "*Din carnetul unui informatician*", autor Mihai Varia, apărut la Editura Promedia, Cluj 1996.

Ascultându-vă, îmi imaginez că cititorii vor comenta sarcastic: "Evident, au trecut o grămadă de ani..." Dar pe mulți dintre foștii elevi i-ați reîntâlnit pe băncile facultății, pe alții la Institutul de Tehnică de Calcul unde ați fost director timp de 22 ani, iar nu în ultimul rând în viața economică a Clujului. Personal nu pot să merg la o expoziție de tehnică de calcul, la o firmă din Cluj, fără să întâlnesc peste tot foști elevi ai Liceul de Informatică în funcții de conducere și/sau specialiști apreciați. Și toate astea în condițiile în care foarte mulți dintre ei azi lucrează în străinătate. Se pune întrebarea: ar fi devenit ei oricum ce sunt azi, sau a contribuit cu ceva și școala?

Sigur că a contribuit, și a contribuit foarte mult. Îmi aduc aminte de perioada când am fost director la ITC și aveam nevoie de programatori buni. Am folosit de multe ori strategia de a angaja absolvenți ai liceului de informatică care se integrau imediat în echipele de dezvoltare software cu rezultate foarte bune și în același timp îi îndemnam, uneori chiar îi obligam să urmeze învățământul universitar seral sau fără frecvență. În 6-7 ani deveneau șefi de proiecte software. Liceele de informatică au devenit re-

și Informatică a Universității Babeș-Bolyai din Cluj. De fapt cred că simțiți și vă deranjează complexul de superioritate față de anumite discipline, transmis și studenților, printre acestea numărându-se practica pedagogică și chiar unele discipline de informatică. Acesta este un obicei mai vechi al învățământului universitar. Trebuie să ne debarasăm rapid de acest complex, pentru că nu poți fi un profesor bun dacă nu ai vocație, iar vocația de dascăl se dezvoltă prin multă practică pedagogică și prin acumulare de experiență "împrumutată" de la persoane care o au.

Am lăsat la urmă întrebarea, al cărei răspuns mă interesează cel mai mult. În cadrul reformei, Ministerul Educației Naționale a dat publicității noul Plan de Învățământ conform căruia liceele teoretice vor funcționa cu două profiluri: real și uman. Profilul real va avea două specialități: Științe ale naturii și Matematică-Informatică. Pe lângă bucuria firească de a constata că informatica în sfârșit a devenit profil (deci știință), constatăm că în planul de învățământ există doar 4-5 ore de informatică pe săptămână. Ar exista o alternativă conform căreia liceele care ar dori să rămână de "informatică" în întregime, să treacă în rețeaua liceelor tehnice. Aș fi interesată să aflu părerea Dvs. despre această alternativă. Ce soluție ați susține în cazul în care vi s-ar solicita părerea?

Susțin soluția propusă de Ministerul Educației Naționale, adică existența în liceele teoretice (profil real) a celor două specialități: *Științele naturii* și *Matematică-Informatică*. Cred că este necesar să existe o diferențiere, chiar din liceu, între cultura generală a unui absolvent care se orientează spre o facultate din domeniul științelor tehnice, a matematicii, a informaticii, a științelor economice și cultura generală a elevului orientat spre medicină, biologie, chimie, în general spre științe ale naturii. Este necesar ca pentru profilul matematică-informatică să se realizeze o pregătire mai aprofundată la disciplinele de matematică și informatică față de disciplinele de biologie și chimie, de exemplu. Însă trebuie bine înțeles că un absolvent de liceu teoretic *este pregătit pentru a urma studii universitare*, dar nu este pregătit și nu primește o calificare pentru a profesa imediat o meserie. În particular, absolventul de la profilul matematică-informatică, după părerea mea, nu va avea aceeași pregătire ca și absolvenții liceelor de informatică de azi. Ei nu vor fi *specialiști* (planul de învățământ prevede numai 3 ore de informatică obligatorii și 2 ore opționale pe săptămână) deci nu vor putea lucra ca asistenți programatori în firme de specialitate.

Ne confruntăm cu următoarele probleme:

1. Trebuie asigurată pregătirea tuturor tinerilor în utilizarea instrumentarului tehnologiei informației la nivel de cultură generală;
2. Trebuie pregătiți specialiști cu studii medii în informatică, cu posibilitatea continuării studiilor la nivel superior.
3. Trebuie hotărâtă soarta actualelor licee și clase de informatică.

Problema pregătirii specialiștilor cu studii medii în tehnologia informației respectiv în informatică și problema actualelor licee de informatică necesită o analiză mai aprofundată și o *corelare cu învățământul universitar de informatică*. Există, după părerea mea, cel puțin două alternative:

1. ***Pregătirea specialiștilor cu studii medii în învățământul universitar prin colegiile de tehnologia informației.*** Această soluție funcționează în prezent în multe orașe și numărul colegiilor de tehnologia informației (durata studiilor - 3 ani) este în creștere. Această variantă ar trebui dusă până la capăt, în sensul de a avea un învățământ universitar de informatică de 5 ani, cu două trepte: prima de 3 ani (colegiul de tehnologia informației) și a doua de 2 ani (studii superioare de informatică). Consider că această structură este cea mai eficientă, și cea mai puțin costisitoare. Modelul este practicat în multe țări în care există un învățământ bun de informatică. *În această variantă actualele licee de informatică ar putea deveni licee teoretice reale cu profil matematică-informatică, asigurând cunoștințe de bază în domeniu.* În altă ordine de idei, trebuie găsită și finalitatea absolut necesară absolvenților de colegiu. Cred că cei care nu vor continua studiile pe durata celor 2 ani în scopul specializării la nivel superior, ar putea deveni profesori de tehnologia informației în școli generale și licee având alt profil decât matematică-informatică (absolvind, evident și *seminarul pedagogic*) sau ar putea practica meseria de asistent informatician în diverse firme de informatică.

2. ***Pregătirea specialiștilor cu studii medii în învățământul preuniversitar și în cel universitar.*** Aceasta este situația actuală din România, unde funcționează liceele de informatică și colegiile universitare de tehnologia informației. Este o situație de compromis, rezultată din necesitatea *de a continua ceea ce este bun în învățământul liceal de informatică*, și necesitatea *de adaptare la structuri noi*, dovedite eficiente - colegiile universitare de tehnologia informației. În această situație, dacă analiza de oportunitate găsește necesară o asemenea soluție, atunci *liceele de informatică ar trebui să devină licee industriale de tehnologia informației*, cu planuri de învățământ și programe analitice adaptate actualei piețe a forței de muncă din acest domeniu. Aceasta înseamnă o orientare mai puternică spre birotică, marketing în informatică, servicii etc. și mai puțin către algoritmică și programare.

Închei cu speranța că răspunsurile Dvs. le vor citi și cei care vor lua decizii în privința viitorului învățământului de informatică liceal și universitar și vor ține cont de sfaturile Dvs.

Vă mulțumesc și vă doresc succes!

Domnul profesor Emil Muntean este cadru didactic la Universitatea "Babeș-Bolyai" din Cluj, poate fi contactat prin e-mail la adresa:
muntean@cs.ubbcluj.ro

ALEGERI prezidențiale

Mihai Scorțaru

S-a spus deja de prea multe ori: "fără calculatoare omenirea ar fi în plop!", dar la fel de multe ori s-a precizat că "totuși, nu este nevoie de atâția algoritmi". E adevărat, nu avem alegeri prezidențiale zilnic...

Alegerile electorale și informatica

Probabil că majoritatea cititorilor (cel puțin cei cu drept de vot) cunosc modul în care se desfășoară primul tur de scrutin al unor alegeri în care dintr-un număr de candidați unul singur va fi ales într-o anumită funcție. Pentru cei care nu au încă drept de vot sau din diferite alte motive nu cunosc acest sistem de votare vom face o scurtă și simplă descriere: fiecare alegător optează pentru unul dintre candidați, iar dacă un candidat obține mai mult de jumătate din voturi atunci el va ocupa funcția respectivă. În caz contrar va avea loc un al doilea tur de scrutin la care se vor califica primii doi candidați în ordinea numărului de voturi obținute la primul tur de scrutin. "Cunoscătorii" știu că procedura este de fapt mai complicată, în realitate ținându-se seama și de alți factori, cum ar fi voturile anulate, prezența la vot etc.

La un moment dat cuiva i-a venit ideea să folosească un calculator pentru a putea anunța mai rapid rezultatele primului tur, adică numele candidatului ales sau "extraordinara" veste că va mai avea loc un tur de scrutin. Se pare că tot mai mulți "oameni importanți" sunt interesați de aceasta și ca urmare o problemă asemănătoare a fost propusă spre rezolvare la etapa regională a concursului ACM din acest an.

În zilele noastre rezolvarea practică a acestei probleme ar fi simplă, deoarece nu vom întâlni un număr foarte mare de candidați pentru un post. Dificultatea va apărea peste puțin timp când, să zicem în jurul anului 2457, sistemele de comunicație vor fi atât de dezvoltate încât oricine va putea candida cu șanse relativ egale la un anumit post. Fiecare alegător va avea atunci voie să voteze pentru oricine și anunțarea rezultatului final va fi o problemă mult mai dificilă.

Enunțul problemei

Dându-se un șir de n *numere care pot lua valori între 1 și* n*, precizați dacă există un număr care apare în șir de mai mult de* n/2 *ori și în caz că acesta există, precizați care este acest număr.*

Am numerotat locuitorii cu drept de vot cu numere între 1 și n, elementul cu numărul i din șir conține votul locuitorului cu numărul i și alegătorul poate opta pentru oricine (valorile elementelor sunt între 1 și n).

Situația este diferită în funcție de numărul alegătorilor alegerile pentru funcția de primar al unui oraș cu 25.000 de locuitori diferind de alegerile pentru funcția de președinte al României la care vor participa peste 10 milioane de alegători. Diferența esențială va fi în păstrarea datelor în memorie, deoarece, deși există posibilitatea memorării unui șir cu 25.000 de elemente, nu același lucru se poate spune despre unul cu peste 10 milioane de elemente (poate în 2457, dar până atunci va crește și numărul alegătorilor).

Vom prezenta câțiva algoritmi pentru cele două cazuri amintite anterior cu scopul de a ilustra diferite tehnici de memorare a datelor, dar și pentru a arăta că deși implementarea anumitor algoritmi este mai dificilă, efortul depus nu este în zadar, deoarece timpul necesar executării programului va fi redus substanțial.

Alegeri locale

Să considerăm pentru început cazul orașului cu cel mult 25.000 de locuitori cu drept de vot.

Cel mai simplu algoritm ar fi ca pentru fiecare element al șirului să contorizăm numărul de apariții în șir și în cazul în care acest număr este mai mare decât n/2 să afișăm fericitul câștigător.

```
Program Alegeri_25000_1;
var id_cand:array[1..25000] of Integer;
    i,j,n,nr:Integer;
    f:Text;
Begin
  Assign(f,'ALEGERI.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do Read(f,id_cand[i]);
  Close(f);
  for i:=1 to n div 2 + 1 do
```

```
  begin
    nr:=1;
    for j:=i+1 to n do
      if id_cand[i]=id_cand[j] then Inc(nr);
    if nr>n div 2
    then
    begin
      Writeln('Primar: ',id_cand[i]); Halt
    end
  end;
  Writeln('Ne vedem duminica viitoare!!!')
End.
```

Am numărat voturile începând doar cu poziția i+1 pentru că dacă posibilul primar ar fi obținut voturi înainte, atunci programul s-ar fi oprit în cazul în care acesta ar fi fost primarul ales în pașii anteriori. Motivul pentru care ciclul controlat de i merge până la valoarea n **div** 2 + 1 este că pentru valori mai mari vom putea aduna mai puțin de jumătate din voturi.

Este evident că ordinul de complexitate al acestui algoritm este $O(n^2)$, iar pentru valoarea n=25.000 timpul de execuție ar fi destul de mare.

Un algoritm mai bun ar fi să păstrăm pentru fiecare candidat numărul voturilor obținute și să verificăm la sfârșit dacă vreun candidat a obținut mai mult de n/2 voturi.

```
Program Alegeri_25000_2;
var frecv_id_cand:array[1..25000] of Integer;
    i,n,id_cand:Integer;
    f:Text;
Begin
  Assign(f,'ALEGERI.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do
  begin
    Read(f,id_cand);
    Inc(frecv_id_cand[id_cand])
  end;
  Close(f);
  for i:=1 to n do
   if frecv_id_cand[i]>n div 2
   then begin Writeln('Primar: ',i); Halt end;
  Writeln('Ne vedem duminica viitoare!!!')
End.
```

Acest algoritm este performant în cazul unui număr mic de alegători, complexitatea lui fiind de ordinul $O(n)$. Implementarea ar putea fi optimizată reducând căutarea câștigătorului doar printre candidații care au primit voturi.

O altă posibilitate care va fi cu siguranță agreată de fanii algoritmilor probabiliști este alegerea la întâmplare a unui candidat votat și verificarea numărului de voturi obținute. În cazul în care primarul a fost ales încă din primul tur de scrutin, la fiecare alegere avem cel puțin 50% șanse să nimerim adevăratul primar. Dacă vom repeta operația de mai multe ori, șansele cresc considerabil. Un calcul foarte ușor arată că pentru n verificări, probabilitatea de a nu găsi primarul, dacă acesta a fost ales din primul tur, este de sub $1/2^n$. Pentru valori ale lui n suficient de mari această valoare poate fi ignorată, de exemplu pentru n=50 șansa de a greși este una din peste un milion de miliarde, adică practic 0. Dacă după 50 de încercări nu am găsit primarul, putem trage concluzia că acesta nu a fost ales din primul tur. Am putea să ne mulțumim și cu numai 20 de încercări sau chiar numai cu 10 deoarece probabilitățile de eroare ar fi mai mici decât 1/1.000.000, respectiv 1/1.000.

```
Program Alegeri_25000_3;
const incercari=50;
var id_cand:array[1..25000] of Integer;
    i,j,n,nr,vot:Integer;
    f:Text;
Begin
  Randomize;
  Assign(f,'ALEGERI.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do Read(f,id_cand[i]);
  Close(f);
  for i:=1 to incercari do
  begin
    vot:=id_cand[Random(n)+1]; nr:=0;
    for j:=1 to n do
      if id_cand[j]=vot then Inc(nr);
    if nr>n div 2
    then
    begin Writeln('Primar: ',vot); Halt end
  end;
  Writeln('Ne vedem duminica viitoare!!!')
End.
```

Alegeri generale

Să trecem acum la lucruri mai importante cum ar fi alegerea președintelui României.

Vom încerca să adaptăm algoritmii prezentați anterior la noua situație (număr mult mai mare de alegători). Este evident că nu avem suficientă memorie convențională pentru a memora niște vectori corespunzători celor folosiți în programele descrise anterior. De aceea, pentru a putea păstra scheletul algoritmilor, va trebui să memorăm datele pe hard-disk. Trebuie să precizăm că de data aceasta numărul de voturi obținute de un candidat nu va mai putea fi reprezentat pe 2 bytes deoarece valorile ar putea depăși 2^{16} și ca urmare suntem nevoiți să folosim o reprezentare pe 4 bytes. Cum pentru fiecare locuitor cu drept de vot vom avea nevoie de 4 bytes, necesarul total de memorie liberă pe disk va fi de $4n$ bytes, unde n este numărul alegătorilor, deci peste 40 MB pentru cei 10 milioane de alegători care vor participa la alegerea președintelui. Pentru a rula în bune condiții programele care vor urma, veți avea nevoie, pe lângă cei 40 MB auxiliari, de încă aproximativ 80 - 90 MB

pentru a putea păstra fișierul de intrare care constă din peste 10 milioane de numere a câte 7 cifre separate de cel puțin un spațiu. În concluzie vă recomandăm să aveți aproximativ 125 MB spațiu liber pe disc înainte să rulați următoarele programe, cu date de test reprezentative.

Primul algoritm prezentat poate fi adaptat astfel:

```
Program Alegeri_10000000_1;
var i,j,n,nr,vot,aux:Longint; f:Text;
Begin
  Assign(f,'ALEGERI.IN'); Reset(f);
  Readln(f,n); Close(f);
  for i:=1 to n div 2+1 do
  begin
    Reset(f); Readln(f);
    for j:=1 to i-1 do Read(f,aux);
    Read(f,vot); nr:=1;
    for j:=i+1 to n do
    begin Read(f,aux);
          if aux=vot then Inc(nr)
    end;
    if nr>n div 2
    then begin Writeln('Presedinte: ',vot);
               Close(f); Halt
         end;
    Close(f)
  end;
  Writeln('Ne vedem duminica viitoare!!!')
End.
```

Se observă că nu a fost nevoie de folosirea celor 40 MB de memorie pentru fișierul auxiliar deoarece nu a fost nevoie de memorarea altor date în afara celor inițiale.

Pentru a adapta al doilea algoritm la această nouă situație va trebui să folosim fișierul auxiliar deoarece va trebui să numărăm voturile obținute de fiecare candidat. Vom inițializa cu 0 numărul de voturi pentru fiecare candidat.

```
Program Alegeri_10000000_2;
var i,n,v,vot:Longint;
    f:Text; g:file of Longint;
Begin
  Assign(f,'ALEGERI.IN'); Reset(f);
  Readln(f,n);
  Assign(g,'ALEGERI.VOT'); Rewrite(g); v:= 0;
  for i:=1 to n do Write(g,v);
  for i:=1 to n do
  begin
    Read(f,vot);
    Seek(g,vot-1); Read(g,v);
    Seek(g,vot-1); Inc(v); Write(g,v)
  end;
  Close(f); Close(g); Reset(g);
  for i:=1 to n do
  begin
    Read(g,v);
    if v>n div 2
    then begin Writeln('Presedinte: ', i);
               Close(g); Halt
         end
  end;
  Close(g);
  Writeln('Ne vedem duminica viitoare!!!')
End.
```

Pentru implementarea algoritmului probabilist pentru cazul alegerilor prezidențiale vom folosi din nou un fișier auxiliar în care vom păstra voturile fiecărui alegător pentru a nu fi nevoie să parcurgem secvențial fișierul de intrare atunci când vom vedea cu cine a votat alegătorul care va fi ales la întâmplare. Deoarece pentru fiecare încercare vom citi de pe disc 10 milioane de numere, vom reduce numărul de încercări de la 50 la 10 deoarece șansele de apariție a erorilor sunt oricum foarte mici.

```
Program Alegeri_10000000_3;
const incercari=10;
var i,j,n,nr,v,vot:Longint;
    g:file of Longint; f:Text;
Begin
  Randomize;
  Assign(f,'ALEGERI.IN'); Reset(f);
  Assign(g,'ALEGERI.VOT'); Rewrite(g);
  Readln(f,n);
  for i:=1 to n do
  begin Read(f,vot); Write(g,vot) end;
  Close(f);
  for i:=1 to incercari do
  begin
    Seek(g,Trunc(n*Random));
    Read(g,vot); Seek(g,0); nr:=0;
    for j:=1 to n do
    begin Read(g,v);
          if v=vot then Inc(nr)
    end;
    if nr>n div 2
    then begin Writeln('Presedinte: ',vot);
               Close(g); Halt
         end
  end;
  Close(g);
  Writeln('Ne vedem duminica viitoare!!!')
end.
```

Pentru a alege aleator un alegător nu am mai putut folosi funcția Random cu parametru un număr întreg deoarece parametrul ar fi trebuit să fi o variabilă de tip Word ceea ce nu ar fi fost posibil în cazul în care valorile ar depăși 2^{16}.

Timpul de execuție al primului algoritm este prea mare pentru ca acest algoritm să poată fi folosit, o apreciere aproximativă demonstrând că ar fi fost nevoie de peste 1000 de zile până când ar fi putut fi anunțate rezultatele,

adică aproximativ la următoarele alegeri. Următorul tabel ilustrează timpii de execuție pentru ceilalți doi algoritmi în diferite cazuri. (Programele au fost rulate pe un calculator 486DX4-100MHz.)

Alegători	%	Algoritm 2 (minute)	Algoritm 3 (minute)
1.000.000	<50	11	24
2.000.000	<50	22	48
5.000.000	<50	56	120
10.000.000	90	49-75	30-90%;45-9%;60-0.9%;75-0,09%
10.000.000	75	49-75	30-75%;45-18.75%;60-4,6875%
10.000.000	>50	49-75	30-50%;45-25%;60-12,5%;75-6,25%
10.000.000	<50	75	165

Se observă că în cazul algoritmului probabilist există o anumită probabilitate pentru un timp de execuție în funcție de procentajul celor care au votat pentru președintele care a fost ales. Timpii de execuție sunt destul de mari, nefiind acceptabili în cazul în care publicul ar dori să afle rezultatele alegerilor imediat după închiderea urnelor.

De aceea trebuie căutat un algoritm care să aibă performanțe superioare. Algoritmul propus este următorul:

- se inițilalizează un contor cu valoarea 1 și primul posibil președinte va fi considerat cel votat de primul alegător
- pentru fiecare vot, în cazul în care acesta este pentru actualul posibil președinte, atunci contorul va fi mărit cu o unitate; în caz contrar acesta va fi scăzut cu o unitate
- dacă contorul a ajuns la 0, posibilul președinte va deveni cel pentru care a votat alegătorul la care am ajuns în acel moment și contorul va fi setat pe valoarea 1
- în final, dacă președintele a fost ales, atunci sigur este cel care a fost considerat ca posibil președinte; pentru a verifica acest lucru va trebui să numărăm voturile acestuia și în cazul în care a obținut mai mult de jumătate din voturi atunci putem afirma că este președintele nou ales.

Probabil că puțini dintre dumneavoastră ar accepta un astfel de algoritm fără o demonstrație. Putem să vă spunem că există o demonstrație matematică riguroasă, dar diferitele semne matematice care ar apărea în cazul în care vom prezenta o astfel de demonstrație v-ar determina să nu mai citiți ceea ce va urma, adică cea mai importantă parte a articolului. Ca urmare vă vom prezenta o demonstrație intuitivă. Dacă posibilul președinte la un moment dat nu este cel care a obținut numărul cel mai mare de voturi atunci, de fiecare dată când va apărea un vot pentru adevăratul președinte contorul va fi scăzut cu o unitate. Datorită faptului că am face mai mult de `n/2` scăderi contorul va ajunge în mod inevitabil o dată la 0 și adevăratul președinte va ajunge să fie considerat ca posibil președinte. Este posibil ca această postură să fie pierdută, dar ea va fi recâștigată din aceleași motive.

Implementarea acestui algoritm presupune parcurgerea de două ori a fișierului de intrare deoarece nu există altă modalitate de a verifica dacă cel care este posibil președinte în final a obținut mai mult de jumătate din voturi.

```
Program Alegeri_10000000_4;
var i,n,nr,vot,contor,presedinte:Longint;
    f:Text;
Begin
  Assign(f,'ALEGERI.IN'); Reset(f);
  Readln(f,n); Read(f,presedinte); contor:=1;
  for i:=2 to n do
  begin
    Read(f,vot);
    if vot=presedinte then Inc(contor)
                      else Dec(contor);
    if contor=0
    then begin presedinte:=vot; contor:=1 end
  end;
  Close(f); nr:= 0; Reset(f); Readln(f);
  for i:=1 to n do
  begin
    Read(f,vot);
    if vot=presedinte then Inc(nr)
  end;
  Close(f);
  if nr>n div 2
  then Writeln('Presedinte: ',presedinte)
  else Writeln('Ne vedem duminica viitoare!')
end.
```

Acest algoritm nu depinde de diferite caracteristici particulare ale datelor de intrare, de aceea timpii de execuție vor depinde numai de valoarea lui n.

Alegători	Timp de execuție (secunde)	Alegători	Timp de execuție (secunde)
1.000.000	15	6.000.000	101
2.000.000	30	7.000.000	118
3.000.000	45	8.000.000	135
4.000.000	60	9.000.000	152
5.000.000	79	10.000.000	168

Din tabel se observă că timpii de execuție variază aproape liniar cu numărul de alegători. Acest lucru era de așteptat deoarece algoritmul prezentat are ordinul de complexitate $O(n)$.

Se observă că timpul de execuție s-a redus de cel puțin 50 de ori, iar intervalul de timp care ar trece de la închiderea urnelor până la anunțarea rezultatelor este foarte mic. Se mai observă că nu este nevoie nici de folosirea unui fișier auxiliar deoarece nu mai avem nevoie de spațiu de memorie suplimentar pentru a memora diferite date.

Acest algoritm poate fi folosit și în cazul alegerilor locale, atunci când voturile sunt memorate într-un șir, dar performanțele sale nu ar fi cu mult mai bune decât cele ale celorlalți doi algoritmi liniari prezentați.

DERANDOMIZARE

Cornelius Croitoru

Primul algoritm probabilist a fost publicat în Gaxeta de Informatică (nr. 2/1997). Domnul profesor Horia Georgescu a "îndrăznit" să atragă atenția rezolvitorilor de probleme asupra acestor algoritmi tocmai din motivul că numerele generate aleator au fost "excluse" de la concursuri mult timp.

Introducere

Se spune că în anii '80 s-a descoperit "aruncarea banului" ca metodă de proiectare a algoritmilor eficienți. Ideea de bază este simplă: se construiește un algoritm probabilist pentru rezolvarea unei probleme, se analizează complexitatea acestuia, după care se face *derandomizarea* - se înlocuiesc generările de numere aleatoare cu apeluri ale unor proceduri de selecție, astfel încât să nu fie afectate concluziile asupra complexității și corectitudinii.

În această notă vom exemplifica metoda de mai sus, considerând unul dintre primii algoritmi de acest tip: *algoritmul lui Luby pentru determinarea unei mulțimi stabile maximale într-un graf.*

Dacă $G = (V, E)$ este un graf, o *mulțime stabilă* în G este o mulțime de vârfuri $S \subseteq V$ astfel încât $\forall\ v, w \in S$, $(v,w) \notin E$. O mulțime stabilă este *maximală* dacă nu este inclusă într-o altă mulțime stabilă. Altfel spus, $S \subseteq V$ este mulțime stabilă maximală dacă:

1. $\forall v,w \in S$: $(v,w) \notin E$ (S este stabilă);
2. $\forall v \in V\text{-}S$, $\exists w \in S$: $(v,w) \in E$ (maximală în raport cu incluziunea).

Considerăm următoarea problemă (*SM*):

Dat fiind graful $G = (V, E)$, să se determine o mulțime stabilă maximală.

Evident, dacă $|V| = n$ și $|E| = m$ atunci un algoritm secvențial de rezolvare a problemei *SM* cu complexitatea $O(n^2)$ este: se parcurge lista vârfurilor și se construiește mulțimea S (inițial vidă), testând la fiecare pas dacă vârful curent poate fi adăugat la S (pentru a forma o mulțime stabilă). Dacă G este reprezentat cu ajutorul listelor de adiacență, atunci o ordonare tip "*min-adjacency*" a mulțimii vârfurilor oferă o soluție de complexitate $O(n+m)$.

O întrebare naturală, care s-a pus la începutul anilor '80, a fost aceea dacă există un algoritm paralel eficient pentru rezolvarea problemei *SM*. Mai precis, considerăm că dispunem de, un număr polinomial în n, procesoare și vrem să le punem să *coopereze* pentru construirea mulțimii stabile maximale în timp polinomial în $\log n$. Modelul de calculator paralel este mai puțin important pentru discuția care urmează, ceea ce urmărim este să punem în evidență dificultatea proiectării unui algoritm paralel.

Vom considera modelul *PRAM* (*Parallel Random Access Machines*): o colecție de procesoare (RAM-uri) independente, sincrone și cu acces la o memorie partajată. Deși nerealist, acest model este util în studiul asimptotic al complexității algoritmilor paraleli.

Clasa de probleme *NC* (*Nick Class*) este clasa problemelor pentru care există un algoritm *PRAM*, care utilizând un număr polinomial de procesoare (în dimensiunea structurală a problemei) rezolvă problema în timp paralel polilogaritmic (polinom în logaritmul dimensiunii structurale).

Întrebarea asupra existenței unui algoritm eficient paralel pentru *SM* se poate formula acum simplu: $SM \in NC$?

Un prim răspuns pozitiv la această întrebare a fost dat de *Karp* și *Wigderson* [1]. O altă demonstrație pentru apartenența lui *SM* la *NC* a fost dată de *Luby* [2]. Soluția lui Luby reprezintă un prototip pentru metodele de derandomizare și merită a fi discutată.

Algoritmul PRAM probabilist al lui Luby pentru rezolvarea problemei *SM*

Input: $G = (V, E)$ graf cu $n = |V|$ vârfuri și $m = |E|$ muchii.
Output: $\Im \subseteq V$, $\Im$ mulțime stabilă maximală în G.

```
Begin
  ℑ:=∅
  while V<>∅ do
  begin
    S:=∅;
    forall v∈V in parallel do
    if d(v)=0
    then S:=S∪{v}
    else se introduce v în S cu probabilitatea 1/(2d(v));
```

```
    forall e∈E in parallel do
    if ambele extremități ale lui e sunt în S
    then elimină din S pe cea cu gradul mai mic (în caz
         de egalitate a gradelor, se elimină una din ele)
    I:=S; { I este o mulțime stabilă în graful curent.}
    ℑ:=ℑ∪I;
    G:=G-(I∪N(I)) { se șterg vârfurile din I∪N(I), }
{ muchiile care au cel puțin o extremitate în I sau N(I) 4 }
  end
End.
```

Este clar că mulțimea I construită într-o iterație **while** este o mulțime stabilă în graful curent G. Mulțimea finală $\mathfrak{I}$ este stabilă, deoarece în fiecare iterație se șterg vârfurile din $N(I)$.

Timpul paralel de executare al fiecărei iterații este $O(1)$. Evident, operațiile cu mulțimi, precum și construirea implicită a grafurilor curente se fac cu ajutorul unor instrucțiuni **forall** cu timp constant de executare.

Dacă se demonstrează că numărul mediu de iterații **while** este de $O(\log n)$ atunci rezultă că întreg algoritmul are timpul mediu paralel $O(\log n)$ (și deci, avem o demonstrație pentru *SM∈RNC*). Nu avem nevoie de o demonstrație completă pentru această afirmație (deși ea se poate deduce cu ușurință), singurul fapt ce trebuie demonstrat este acela că la fiecare iterație numărul mediu de muchii ce se șterg din graful curent este cel puțin o fracție constantă din numărul de muchii ale acestuia. Atunci când vom face derandomizarea, vom obține aceeași proprietate pentru numărul de muchii șterse, și deci, în acest ultim caz, numărul iterațiilor va fi $O(\log n)$.

În cele ce urmează vom demonstra că în fiecare iterație se șterg în medie măcar 1/72 din numărul total de muchii ale grafului curent (cel cu care se începe iterația) $G=(V, E)$.

Lema 1. Pentru orice vârf $v \in V$, $\Pr(v \in I) \geq \frac{1}{4d(v)}$.

Demonstrație:

$$\Pr(c \in I)=\Pr(v \in I \mid v \in S)\cdot \Pr(v \in S)=\frac{1}{2d(v)}\cdot \Pr(v \in I \mid v \in S)=$$

$$=\frac{1}{2d(v)}.[1-\Pr(v \notin I \mid v \in S)]$$

Un vârf $v \in S$ nu va aparține lui I dacă $\exists u \in N(v)$ cu $d(u) \geq d(v)$ și $u \in S$.

Fie $L(v)=\{u \mid u \in N(v),\ d(u) \geq d(v)\}$

Atunci $\Pr(v \notin I \mid v \in S) \leq \Pr(\exists\, u \in L(v) \cap S \mid v \in S) \leq$

$$\leq \sum_{u \in L(v)} \Pr(u \in S \mid v \in S)$$

"aruncări independente 2 câte 2"

$$\leq \sum_{u \in L(v)} \Pr(u \in S)=\sum_{u \in L(v)} \frac{1}{2d(u)} \leq \sum_{u \in L(v)} \frac{1}{2d(v)} \leq \frac{d(v)}{2d(v)}=\frac{1}{2}$$

Deci:

$$\Pr(v \in I)\frac{1}{2d(v)}[1-\Pr(v \notin I \mid v \in S)] \geq \frac{1}{2d(v)}\left(1-\frac{1}{2}\right)=\frac{1}{2d(v)}$$

Definiție:

Un vârf $v \in V$ se numește *bun* dacă $\sum_{u \in N(v)} \frac{1}{2d(u)} \geq \frac{1}{36}$.

O muchie este *bună* dacă cel puțin unul din vârfurile pe care le unește este *bun*. Vârfurile sau muchiile care nu sunt bune se numesc *rele*.

Lema 2. Dacă v este un vârf bun atunci $\Pr(v \in N(I)) \geq \frac{1}{36}$.

Consecință:

Probabilitatea de a șterge o muchie bună este cel puțin 1/36 (este cel puțin egală cu probabilitatea ca una din extremitățile sale - care este vârf bun - să aparțină lui $N(I)$).

Demonstrație:

Fie v un vârf bun.

Cazul 1. $\exists u \in N(v)$ astfel încât $d(u) \in \{1,2\}$.

Atunci $\Pr(v \in N(I)) \geq \Pr(u \in I) \geq \frac{1}{4d(u)} \geq \frac{1}{8} > \frac{1}{36}$

Cazul 2. $u \in N(v)$, $d(u) \geq 3$.

Atunci, cum v este bun,

$$\sum_{u \in N(v)} \frac{1}{2d(u)} \geq \frac{1}{6} \text{ și } \forall u \in N(v) \frac{1}{2d(u)} \leq \frac{1}{6}.$$

Există $M(v) \subseteq N(v)$ astfel încât $\frac{1}{6} \leq \sum_{u \in M(v)} \frac{1}{2d(u)} \leq \frac{1}{3}$.

$\Pr(v \in N(I)) \geq \Pr(\exists u \in M(v) \cap I) \geq$

$$\geq \sum_{u \in M(v)} \Pr(u \in I) - \sum_{u,w \in M(v)} \Pr(u \in I \text{ și } w \in I) \geq$$

$$\text{(lema 1)} \sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{\substack{u,w \in M(v) \\ u \neq v}} \Pr(u \in S \text{ și } w \in S) = \text{("independente 2 câte 2")}$$

$$\sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u,w \in M(v)} \Pr(u \in S)\Pr(w \in S) =$$

$$\sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{\substack{u,w \in M(v) \\ u \neq v}} \frac{1}{2d(u)} \frac{1}{2d(w)}$$

$$\left(\sum_{u \in M(v)} \frac{1}{2d(u)}\right)\left[\frac{1}{2} - \sum_{w \in M(v)} \frac{1}{2d(w)}\right] \geq \frac{1}{6} \cdot \left(\frac{1}{2}-\frac{1}{3}\right)=\frac{1}{36}$$

Lema 3. Cel puțin jumătate din muchii sunt bune.

Demonstrație:

Orientăm muchiile grafului astfel: dacă $(u,v) \in E$ o orientăm de la u la v dacă și numai dacă $d_G(u) \leq d_G(v)$. În graful orientat obținut este valabilă următoarea proprietate: numărul arcelor care ies dintr-un vârf rău este cel puțin de două ori mai mare decât al celor care ies.

Fie v rău: $\sum_{u \in N(v)} \frac{1}{2d(u)} < \frac{1}{6}$.

După orientare, $N(v)=A \cup B$

$u \in A \Rightarrow uv$ intră în v $(d(u) \leq d(v))$

$u \in B \Rightarrow vu$ iese din $u(d(v) \le d(w))$

Trebuie să demonstrăm că $|B| > 2|A|$.

Presupunem că $|B| \le 2|A|$ și atunci $3|A| \ge d(v)$, iar

$$\sum_{u \in N(v)} \frac{1}{2d(u)} = \sum_{u \in A} \frac{1}{2d(u)} + \sum_{u \in B} \frac{1}{2d(u)} \ge \sum_{u \in A} \frac{1}{2d(u)} \ge$$

$$\ge \sum_{u \in A} \frac{1}{2d(v)} \ge \frac{|A|}{2d(v)} \ge \frac{d(v)}{3 \cdot 2d(v)} = \frac{1}{6}.$$

Am obținut că v este bun, contradicție.

Folosind proprietatea pusă în evidență putem asocia fiecărui arc rău (provenit din orientarea unei muchii rele) o mulțime de arce ce ies din extremitatea sa finală, asocierea fiind astfel încât la arce distincte rele corespund mulțimi disjuncte. Rezultă că există de două ori mai multe arce în total decât cele rele, deci numărul arcelor rele este cel mult jumătate din numărul total de arce, deci numărul muchiilor bune este măcar jumătate din numărul total de muchii.

Teoremă:

Numărul mediu de muchii care se șterg la fiecare iterație a algoritmului 1 este mai mare sau egal cu $\frac{|E|}{72}$ *(*$|E|$ *este numărul de muchii ale grafului curent).*

Demonstrație:

Fie X variabilă aleatoare reprezentând numărul muchiilor șterse într-o iterație. Pentru fiecare $e \in E$ considerăm X_e variabilă aleatoare

$$X_e = \begin{cases} 1, \text{dacă } e \text{ este stearsă} \\ 0, \text{altfel} \end{cases}$$

Atunci $X = \sum_{e \in E} X_e$ și media lui X este suma mediilor variabilelor aleatoare X_e și $MX = \sum_{e \in E} MX_e \ge \sum_{e \in E} MX_e =$

$$= \sum_{e \in E} Pr(e \text{ stearsă}) \underset{\text{Consec.}}{\ge} \sum_{\substack{e \in E \\ e \text{ bună}}} \frac{1}{36} \underset{\text{lema3}}{\ge} \frac{|E|}{72}$$

Observație:

Se poate demonstra că din această teoremă rezultă că numărul mediu de iterații necesare până se șterg toate muchiile (când algoritmul se termină) este $O(\log m)$ unde m este numărul de muchii ale grafului inițial (acesta este un exercițiu standard în teoria probabilităților discrete). Se obține deci că $SM \in RNC$ (un certificat al acestei apartenențe este algoritmul 1, pentru că $m = O(n^2)$).

Derandomizarea

În fiecare iterație a algoritmului 1 se execută n aruncări, independente două câte două, ale unor monezi "strâmbe".

(Pentru vârful $u \in V$ se consideră o monedă având *Pr* (de apariție a celeilalte fețe) $= 1 - \frac{1}{2d(n)}$).

Vom demonstra că aceste n aruncări se pot genera cu ajutorul unui șir de $O(\log n)$ biți aleatori. Fie p un număr prim astfel încât $n \le p < 2n$. Vârfurile grafului se vor eticheta cu elemente din Z_p. Pentru fiecare $u \in V$ se determină $0 < a_u < p$ astfel încât:

$$\left| \frac{a_u}{p} - \frac{1}{2d(u)} \right| = \min \left\{ \left| \frac{x}{p} - \frac{1}{2d(u)} \right| ; x \in Z_p \right\}$$

Nu se vor genera aruncări cu probabilitățile

$\left(\frac{1}{2d(u)}, 1 - \frac{1}{2d(u)} \right)$ ci cu probabilitățile $\left(\frac{a_u}{p}, 1 - \frac{a_u}{p} \right)$.

Aceasta nu afectează concluzia din teoremă).

Pentru fiecare $u \in V$ se consideră $A_u \subset \{0,1,...,p-1\}$ cu $|A_u| = a_u$.

Se aleg uniform la întâmplare două elemente $x, y \in Z_p$; aceasta necesită $2p = O(\log n)$ biți aleatori. Pentru fiecare $u \in V$ se calculează $x + uy$ în Z_p.

Dacă $x + uy \in A_u$ se consideră că moneda lui u a căzut cu fața în sus și în algoritmul 1 se va introduce u în S.

Trebuie să demonstrăm că $Pr(x + uy \in A_u) = \frac{a_u}{p}$.

Să observăm că pentru orice $z, y \in Z_p$ există un unic $x \in Z_p$ astfel încât $x + uy = z$. Folosind această observație avem:

$$Pr(x + uy \in A_u) = \frac{1}{p^2} \cdot \left| \{ (x,y); x + uy \in A_u \} \right| =$$

$$= \frac{1}{p^2} \sum_{z \in A_u} \left| \{ (x,y) | x + uy = z \} \right| = \frac{1}{p^2} \cdot \sum_{z \in A_u} p = \frac{1}{p^2} \cdot |A_u| \cdot p = \frac{a_u}{p}.$$

De asemenea, trebuie să demonstrăm independența 2 câte 2 a aruncărilor, adică $\forall u \ne v$, $Pr(x+uy \in A_u$ și $x+vy \in A_v) = Pr(x+uy \in A_u) \cdot Pr(x+vy \in A_v)$.

Într-adevăr, $Pr(x+uy \in A_u$ și $x+vy \in A_v) =$

$$= \frac{1}{p^2} \left| \{ (x,y) | x + uy \in A_u \text{ și } x + vy \in A_v \} \right| =$$

$$= \frac{1}{p^2} \sum_{z \in A_u} \sum_{w \in A_v} \left| \{ (x,y) | x + uy = z \text{ și } x + vy = w \} \right|.$$

Să observăm că în corpul Z_p sistemul $\begin{cases} x + uy = z \\ x + vy = w \end{cases}$ are determinantul $v - u \ne 0$ și deci are soluție unică. Rezultă că se poate scrie în continuare $Pr(x+uy \in A_u$ și $x+vy \in A_v) =$

$$= \frac{1}{p^2} \sum_{z \in A_u} \sum_{w \in A_v} 1 = \frac{a_u \cdot a_v}{p^2} = \frac{a_u}{p} \cdot \frac{a_v}{p} = Pr(x+uy \in A_u) \cdot Pr(x+vy \in A_v)$$

Rezultă că dacă se dispune de șirul s de lungime $O(\log n)$ de biți aleatori, algoritmul de generare a celor n aruncări este:

Algoritmul GenA:

Input: s șir de lungime $O(\log n)$ de biți aleatori; mulțimea $S \subseteq V$ a vârfurilor de grad 0.

Output: mulțimea S la care se adaugă vârfurile de grad nenul din V cu o probabilitate "apropiată" de $\frac{1}{2d(u)}$, independente unele de altele.

```
Begin
  Construiește x și y din s;
  Forall u∈V in parallel do
  begin
    Construiește a_u, A_u
    if x+uy∈A_u then S:=S∪{u}
End.
```

Numărul șirurilor de lugime $O(\log n)$ cu componente 0 și 1 este $2^{O(\log n)} = n^{O(1)}$. Fie S mulțimea acestor șiruri.

Rezultă că în loc să generăm un șir de lungime $O(\log n)$ de biți aleatori pentru care să aplicăm algoritmul GenA putem proceda astfel:

```
forall (s∈S) pardo
  aplică o iterație a algoritmului 1 folosind GenA pentru
  a face selecția aleatoare.
```

Fiecare aplicare paralelă va construi o mulțime $I(s)$.

Notăm cu $X(s)$ numărul muchiilor care se șterg din graful curent prin îndepărtarea lui $I(s)$. $X'(s)$, $s \in S$ reprezintă valorile posibile ale variabilei aleatoare X reprezentând numărul de muchii șterse într-o iterație a algoritmului.

Cum $MX \geq \frac{|E|}{72}$ rezultă că $\exists s_0 \in S$ astfel încât:

$$X(s_0) \geq \frac{|E|}{72}.$$

Rezultă că dacă procedăm astfel:

```
forall s∈S pardo
  aplică o iterație a algoritmului 1 folosind GenA și
  determină I(s) și X(s); determină în paralel
```

$$\max_{s \in S} X(s) = X(s_0)$$

atunci $I(s_0)$ va fi o mulțime stabilă construită (în mod determinist) într-o iterație și are proprietatea că dacă se șterge $I(s_0) \cup N(I(s_0))$ din graful curent, atunci numărul muchiilor șterse este $\geq 1/72$ din numărul de muchii ale acestui graf.

Descrierea completă a algoritmului determinist

Input: $G=(V,E)$ graf cu $n = V$ vârfuri și $m = E$ muchii

Output: $\Im \subseteq V$, $\Im$ mulțime stabilă maximală

```
Begin
  ℑ:=∅
  while V<>∅ do
  begin
    S:=∅;
    forall v∈V in parallel do
      if d(v)=0 then S:=S∪{u}; V:=V-{v}
      determină p număr prim |V| ≤ p ≤ 2|V|
    forall (x,y)∈{0,1,...,p-1}×{0,1,...,p-1} in parallel
    do
      begin
        S(x,y):=∅
        forall u∈V in parallel do
        begin
          determină a_u∈{1,...,p-1} a.î.
```

$$\left|\frac{a_u}{p}\ \frac{1}{2d(u)}\right| = \min\left\{\left|\frac{x}{p}\ \frac{1}{2d(u)}\right| : x \in Z_p\right\}$$

```
          construiește A_u.

          if x+uy∈A_u then S(x,y):=S(x,y)∪{u}
        end;
        forall e∈E in parallel do
          if ambele extremități ale lui e sunt în S(x,y)
            then elimină din S(x,y) extremitatea cu
                 gradul mai mic;
        I(x,y):=S(x,y)
        determină G(x,y):=G-(I(x,y)∪N(I(x,y))∪S)
        determină m(x,y) numărul de muchii ale lui
        G(x,y)
      end;
    Determină (x0,y0) astfel încât:
      m(x0,y0)=min{m(x,y); x,y∈Z_p}
    I:=S∪I(x0,y0)
    ℑ:=ℑ∪I
    G:=G(x0,y0)
  end
End.
```

Numărul iterațiilor `while` este $O(\log n)$, fiecare iterație necesită $O(\log n)$ timp paralel. Rezultă că întreg algoritmul se execută în timpul paralel $O(\log^2 n)$. Numărul de procesoare utilizate este $O(n^2)$.

Bibliografie

1. R.M. Karp, A. Wigderson - *A Fast Parallel Algorithm for the Maximal Independent Set Problem*, Proc. 16th Symp. Theory of Computing, p. 266-272, ACM, 1984
2. Luby - *A Simple Parallel Algorithm for the Maximal Independent Set Problem*, Proc. 17th Symp. Theory of Computing, p. 1-10, ACM, 1985

Domnul profesor Cornelius Croitoru este cadru didactic la Universitatea "Al. I. Cuza" din Iași. Poate fi contactat prin e-mail la adresa: croitoru@infoiasi.ro

Problema RUTAJULUI

Viorica Covali

Pe durata activității unui sistem cu prelucrare distribuită, schimbul de informații dintre noduri se realizează prin intermediul canalelor de comunicație. În cazul în care între două noduri nu există un canal direct, se pune problema găsirii unui drum format din canale și noduri care leagă cele două noduri sursă-destinație. De obicei, între două noduri pot exista mai multe drumuri, deci apare problema alegerii unuia prin care se va livra informația. Cum se alege un drum din mulțimea drumurilor posibile? Care sunt criteriile după care se face alegerea? Răspunsurile le oferă rezolvarea problemei rutajului.

Tipuri de comunicații

Sistem de prelucrare distribuită: o colecție de *calculatoare*, *procesoare* sau *procese* autonome (denumite *noduri*) interconectate

Canal de comunicație: legături între calculatoare, care asigură transferul de date

Comutarea de mesaje și pachete

Comutare: acțiune ce deplasează entitățile de informații în rețea de la un nod sursă la un nod destinație

Entități de informație: mesaje și pachete

Mesaj: entitate de informație care nu are o lungime fixă prestabilită

Pachet: entitate de informație cu o lungime fixă, stabilită apriori

Comutare de mesaje: tip de comunicație în care entitatea de transfer este mesajul

Lungimea mesajului poate varia, dar trebuie să existe o lungime maximă. În cazurile în care apar mesaje mai lungi, operațiile de fragmentare și reasamblare revin programatorului.

Definiția problemei

Celor nefamiliarizați cu terminologia sistemelor distribuite le recomandăm să citească mai întâi coloana din dreapta.

În cele ce urmează vom considera că mesajele au lungime fixă prestabilită și nu vom face distincție între mesaj și pachet.

Problema rutajului constă în determinarea drumurilor de comunicație între perechi de noduri ale sistemului distribuit. Pentru rezolvarea acestei probleme, fiecare nod trebuie să conțină informații despre topologia rețelei. Aceste informații sunt grupate în *tabele de rutaj* locale fiecărui nod.

Dacă un nod *i* recepționează un mesaj *m* cu destinația *j*, rutajul, utilizând tabelul de rutaj, precizează spre care vecin *k* al lui *i* va fi transmis mesajul.

Momentul de timp în care rutajul stabilește traseul, depinde de tipul rutajului folosit: *rutaj cu canale virtuale* sau *rutaj dinamic.*

În cazul *rutajului cu canale virtuale*, nodurile prin care va trece mesajul pentru a ajunge la destinație sunt stabilite la inițierea sesiunii de comunicație și toate pachetele acestei sesiuni vor urma aceeași cale.

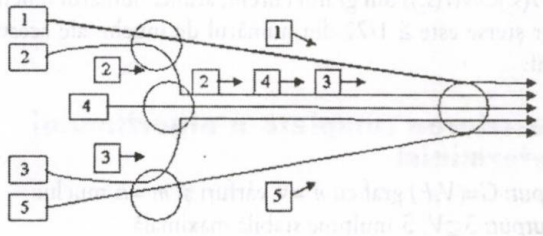


Fig.1 Rutajul în rețea cu canale virtuale

În cazul rutajului dinamic, decizia se ia în timpul parcurgerii drumului spre destinație, în fiecare nod. Prin urmare, pachetele livrate în aceeași sesiune de comunicare pot avea drumuri diferite.

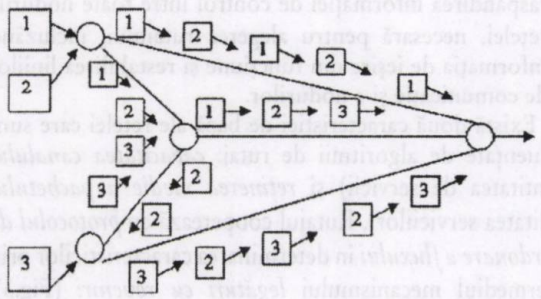


Fig.2 Rutajul dinamic.

Problema rutajului poate fi împărțită în două subprobleme:

1. Determinarea tabelelor de rutaj
   Tabelele de rutaj trebuie să fie calculate la inițializarea rețelei și să fie actualizate la fiecare modificare a topologiei rețelei.
2. Direcționarea pachetului
   La trimiterea unui pachet în rețea, acesta trebuie să fie direcționat, utilizând tabelele de rutaj.

Pentru soluționarea acestor subprobleme, sunt elaborați diverși algoritmi de rutaj.

Algoritmul de rutaj este un protocol al nivelului rețea (al modelului OSI) *care are ca sarcină coordonarea pachetelor în timpul deplasării lor prin rețea, până ajung la destinație.*

Obiectivele în proiectarea unui algoritm de rutaj sunt:
- generarea, pentru fiecare nod a unei proceduri de luare a unei decizii care să efectueze această sarcină;
- garantarea livrării fiecărui pachet trimis în rețea.

În general, pentru îndeplinirea acestor obiective se folosește o colecție de algoritmi a căror complexitate rezultă din următoarele:

1. Rutajul necesită coordonarea funcționării tuturor nodurilor din rețea.
2. Sistemul de rutaj trebuie să finalizeze ieșirile din funcțiune a liniilor de comunicații efectuând redirecționarea fluxului de mesaje și reînnoirea bazelor de date folosite de sistem.
3. Pentru a atinge performanțe mai bune, algoritmii de rutaj pot modifica rutarea în cazul în care unele regiuni ale rețelei devin supraîncărcate.

În cele ce urmează vom prezenta mai multe metode de rutaj în care vor fi analizate două aspecte ale problemei de rutaj:

1) Alegerea traseului pentru a atinge performanțe mai bune:

Comutare de pachete: tip de comunicație în care informația ce urmează a fi livrată este fragmentată în pachete de dimensiune fixă

- Operațiile de fragmentare și reasamblare sunt efectuate de către sistem.
- Fragmentarea informației se efectuează în nodul sursă înainte de livrare.
- Operația de reasamblare este efectuată în nodul destinație prin concatenarea pachetelor ce compun informația inițială.

Observație:
Pe parcursul unei sesiuni de transfer între nodul sursă și nodul destinație, nodurile rețelei partajează în permanență legăturile de comunicație. Astfel, pachetele (mesajele) aceleiași sesiuni de transfer pot avea drumuri diferite.

Comutarea de circuite

Circuit virtual: drum stabilit între două noduri, în comutarea de circuite, înainte de toate comunicațiile

Pachet de control: definește un drum între sursă și destinație pe care se vor deplasa toate pachetele (mesajele) emise și recepționate de aceste noduri pe tot parcursul sesiunii de comunicație. La încheierea comunicațiilor un alt *pachet de control* va suprima circuitul.

Fiecare canal fizic care materializează un astfel de circuit virtual poate fi divizat în tranșe de timp, câte o tranșă pentru fiecare canal virtual (multiplicare temporară: peste același canal fizic trec mai multe circuite virtuale, pentru diferite sesiuni de transmisie).

La stabilirea circuitului virtual, *pachetul de control inițial* rezervă, de asemenea, tranșe temporale peste canalul fizic și în toate nodurile ce compun canalul virtual se stabilește corespondența între tranșele de timp ale canalului de intrare cu ale celui de ieșire.

Important: tranșele temporale care materializează canalele virtuale ale circuitului virtual sunt alocate pe parcursul întregii durate de viață a circuitului virtual, indiferent dacă sunt sau nu utilizate.

Diferențe între comutarea de circuite și comutarea de pachete (mesaje):

a) mesajele aceleiași sesiuni de comunicații parcurg același drum (circuitul virtual), pe când în cazul comutării de pachete ele pot parcurge drumuri diferite;
b) determinarea traseului se face o singură dată la stabilirea circuitului virtual, iar în cazul comutării de pachete canalul de ieșire din nod se stabilește în acest nod. De obicei, în comutarea de pachete, entitățile de transfer sunt îndrumate de nodul în care au sosit spre canale mai puțin aglomerate, evitându-se astfel supraîncărcarea canalelor;
c) canalele fizice (tranșele temporale) sunt alocate static, ceea ce conduce la o utilizare ineficientă a resurselor rețelei, fapt care nu se întâmplă la comutarea de pachete;
d) căderea unui canal fizic care materializează un canal virtual, fragmentează circuitul virtual, ceea ce determină întreruperea comunicației; în cazul comutării de pachete, entitățile de transfer pot alege alte linii de comunicație.

Rutaj cu canale virtuale și rutaj dinamic

Entitățile de transfer (pachet, mesaj) se deplasează în rețea prin nodurile intermediare dintre sursă și destinație pe baza unor informații apărute în momentul alegerii traseului. Momentele de timp în care se ia decizia de a alege traseul determină două tipuri de comunicații.

Rutajul cu canale virtuale: în timpul inițializării sesiunii de comunicație stabilește un drum concret ce se păstrează pe întreaga durată a sesiunii.

Rutajul cu canale virtuale este analog cu comutarea de circuite în sensul că se utilizează un drum fixat pentru deplasarea pachetelor, dar capacitatea canalului este folosită numai în cazul în care este necesar (spre deosebire de comutarea de circuite, care alocă tranșa temporală pentru întreaga durată de viață a circuitului).

Rutajul dinamic: pentru fiecare pachet în parte se determină un drum individual în rețea pe baza informației curente din nod în momentul atingerii lui.

Rutaj: descrie procedura de decizie prin care un nod selecționează unul dintre vecinii lui, spre care va direcționa un pachet, ce trebuie să atingă o destinație finală.

Tabele de rutaj: conțin informații despre topologia rețelei, corespunzător fiecărui nod.

a) elaborarea algoritmilor de rutaj ce determină drumuri de cost minim;
b) elaborarea algoritmilor de rutaj având caracteristici apropiate de cele optime.

2) Al doilea aspect al problemei de rutaj constă în răspândirea informației de control între toate nodurile rețelei, necesară pentru alegerea rutajului, incluzând informația de ieșire din funcțiune și restabilirea liniilor de comunicație și a nodurilor.

Există două caracteristici de bază ale rețelei care sunt influențate de algoritmii de rutaj: *capacitatea canalului* (cantitatea de servicii) și *reținerea medie a pachetului* (calitatea serviciilor). Rutajul cooperează cu *protocolul de coordonare a fluxului* în determinarea caracteristicilor prin intermediul mecanismului *legături cu reveniri* (Fig.3). Există situații, în care cantitatea de informație ce vine din exterior este mai mare decât cea care poate fi deservită, chiar și în cazul unui rutaj optim. Sarcina de a coordona fluxul de informație revine algoritmilor de coordonare a fluxului.

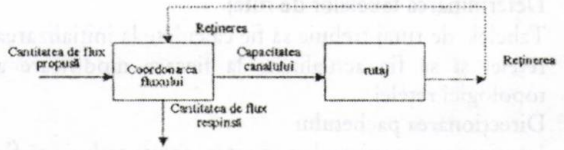


Fig.3 Legătura dintre rutaj și coordonarea fluxului.

Dacă fluxul de informație apărut de la surse din exterior este relativ mic față de capacitatea canalelor, atunci el va fi integral primit de rețea și vom avea:

capacitatea de trecere = fluxul din exterior,

în caz contrar

capacitatea de trecere =
fluxul din exterior - cantitatea de flux respinsă.

Una dintre sarcinile de eficiență ale unui sistem cu prelucrare distribuită este de a efectua un număr mai mare de servicii într-un timp mai redus, atât cererilor din interior cât și celor apărute din exterior. Pe lângă componentele fizice ale sistemului distribuit (canalele de comunicații având capacitate mai mare, noduri cu prelucrare rapidă etc.) o mare importanță o are alegerea tipului de rutaj în rezolvarea acestei sarcini. Prin urmare, dacă algoritmul de rutaj reușește să mențină relativ bine o *reținere mică* a pachetelor în noduri care se găsesc în drum spre destinație, atunci algoritmul de coordonare a fluxului permite intrarea în rețea a unui flux mai mare de informație.

Algoritmi de rutaj de bază

Scopul algoritmilor de rutaj este de a coordona deplasarea mesajelor prin rețea până ajung la destinație. Pentru atingerea unor performanțe mai bune, algoritmii de rutaj au ca scop determinarea unui traseu bun.

Metodele de rutaj folosite în acest scop se bazează pe

un șir de algoritmi de determinare a drumurilor optime în grafe.

Algoritmii de rutaj se deosebesc în funcție de criteriul de optimalitate al traseului:

- ***Număr minim de pași:*** costul drumului se măsoară prin numărul canalelor traversate. Evident, criteriul de optimalitate va fi asigurat de utilizarea drumurilor cu număr minim de canale.
- ***Calea cea mai scurtă:*** fiecărui canal îi este asociată, pe cale statistică, o pondere pozitivă iar costul unui drum se exprimă prin suma ponderilor canalelor din drum. Un algoritm de acest tip utilizează drumul cu ponderea totală minimă.
- ***Întârzieri minime:*** fiecărui canal îi este asociată în mod dinamic o pondere care depinde de traficul pe canalul respectiv. Un algoritm cu întârzieri minime actualizează tabelele de rutaj în mod continuu, astfel încât în orice moment să se aleagă drumul pentru care întârzierile totale sunt minime. Deoarece întârzierile de pe canale se datorează traficului, diferitele pachete transmise pe rețea se influențează reciproc.

Algoritmi de rutaj bazați pe determinarea drumului de cost minim

Elaborarea algoritmilor din această clasă se bazează pe algoritmii de determinare a drumului de cost minim în grafuri, adaptați la context distribuit.

Algoritmii din această clasă trebuie să îndeplinească următoarele criterii:

1. Costul trimiterii unui pachet pe un drum D este independent de gradul de utilizare a acelui drum, în particular de faptul că muchiile din D sunt utilizate de alte mesaje.
2. Costul drumului rezultat din concatenarea a două drumuri este egal cu suma costurilor celor două drumuri. Ca o consecință, costul drumului vid este egal cu 0.
3. Graful asociat rețelei nu conține nici un ciclu de cost negativ.

Algoritmul distribuit și asincron al lui Bellman-Ford

Principiul de bază

Algoritmul propus este o adaptare la *context distribuit* a algoritmului lui *Ford* inspirat din principiul programării dinamice a lui *Bellman*. O caracteristică a acestui algoritm constă în faptul că necesită păstrarea unei cantități mici de informație în nodurile rețelei, deoarece pentru fiecare nod este suficientă informația privitoare la lungimile canalelor ce pleacă de la el și numărul nodurilor din rețea.

Vom considera că lungimea d_{ij} a fiecărei muchii (i,j) este pozitivă și că lungimile lor vor rămâne fixe după momentul de timp t_0, moment până la care au loc toate modificările în rețea.

Principiul pe care se bazează algoritmul lui Bellman-

Tipuri de rutaj

Tipul de rutaj ales influențează într-o mare parte caracteristicile de bază ale unui sistem cu prelucrare distribuită.

În cele ce urmează vom prezenta câteva tipuri de rutaj, precum și criteriile de bază ce le diferențiază.

Rutaj centralizat: alegerea traseului se efectuează de un nod central. Fiecare nod trimite poziția sa în rețea nodului central care calculează tabela de rutaj.

Rutaj distribuit: fiecare nod își calculează propriile tabele de rutaj pe baza informațiilor primite de la nodurile vecine.

O altă clasificare este bazată pe faptul că *se modifică sau nu* rutajul în raport cu intensitatea fluxului de intrare. În acest caz deosebim: rutaj static și rutaj adaptiv.

Rutaj static: traseele se stabilesc o singură dată pentru toate perechile de noduri, la inițializare.

Rutaj adaptiv: noile rute se calculează în funcție de fluxul de informație sau de starea canalelor de comunicație care sunt sau nu în funcțiune.

În cazul *rutajului static*, pentru fiecare pereche de noduri (sursă, destinație) drumul este fixat și nu depinde de oscilațiile fluxului de mesaje. El poate să se modifice numai în cazul ieșirii din funcțiune a unui nod sau a unui canal de comunicație. La folosirea acestui mod de rutaj nu se poate ajunge la o capacitate de trecere mare la variații considerabile ale fluxului de intrare. Această metodă de rutaj este recomandabilă fie pentru sisteme simple, fie în cazul în care eficiența lucrului în rețea este neesențială.

În majoritatea sistemelor cu prelucrare distribuită având ca și tip de comunicație comutarea de pachete este utilizat *rutajul adaptiv* în care drumurile de la nodul sursă la nodul destinație pentru un nou flux se modifică ca urmare a schimbării stării canalului.

În rezultatul modificării fluxului de intrare pe anumite porțiuni ale rețelei pot apărea supraîncărcări. În acest caz algoritmul de rutaj trebuie să încerce modificarea traseelor și să direcționeze fluxul de mesaje, ocolind nodurile aglomerate de pachete.

Ford constă în determinarea distanței minime D_i de la nodul i la nodul destinație comun (notat, de exemplu, prin 1) pentru toate nodurile, cu ajutorul următoarelor formule:

$$D_i^{(h+1)} = \min_{s \in N(i)} [d_{ij} + D_i^{(h)}] \quad (1)$$

$$D_1^{(h+1)} = 0 \quad (2)$$

unde $N(i)$ este mulțimea nodurilor vecine cu i. (În practică pentru fiecare nod destinație se reexecută algoritmul.)

Pentru condițiile inițiale

$$D_i^{(0)} = \infty, i \neq 1 \text{ și } D_1^{(0)} = 0 \quad (3)$$

algoritmul lui Bellman-Ford determină distanțele minime între nodurile rețelei și destinație. Distanțele D_i pot fi determinate distribuit în fiecare nod executându-se iterația (1). Dacă se stabilește o sincronizare, astfel încât nodurile execută concomitent iterația (1) de ordinul h, făcând schimb de rezultate între iterații, atunci pornind cu condițiile inițiale (3), algoritmul termină execuția cel mult în N-1 pași (N - numărul nodurilor în rețea).

Stabilirea unei sincronizări în sisteme cu prelucrare distribuită este dificilă, de aceea se folosește o alternativă a acestui algoritm care funcționează în regim asincron și care nu necesită un protocol de inițializare respectiv protocol de reluare a execuției algoritmului. Strategia de execuție constă în modificarea în timp a valorii lui D_i folosind noua valoare D_j, primită de la vecinii $j \in N(i)$ și ultimele lungimi, precum și starea canalelor adiacente nodului i după formula:

$$D_j = \min_{j \in N(i)} [d_{ij} + D_j] \quad (1^*)$$

Reprezentarea formală a algoritmului

În orice moment t în fiecare nod $i \neq 1$ există:

$D_j^i(t)$ - valoarea distanței minime de la j la nodul 1, $j \in N(i)$, care a fost transmisă nodului i ultima dată;

$D_i(t)$ - valoarea distanței minime de la i la nodul 1, care a fost calculată ultima dată în conformitate cu iterația Bellman-Ford.

Distanțele pentru nodul 1 sunt întotdeauna nule.

$D_1(t)=0$ pentru orice $t \geq t_0$;

$D_1^i(t)=0$ pentru orice $t \geq t_0$ și i, pentru care $1 \in N(i)$.

Fiecare nod i conține lungimile d_{ij} pentru toți $j \in N(i)$, care se presupun a fi pozitive și nu se modifică în timp după momentul t_0. Presupunem că distanțele dintre noduri nu se modifică decât în momentele $t_0, t_1, ...; t_{m+1} > t_m$ pentru toți m și $t_m \to 0$ când $m \to \infty$, în care în fiecare nod $i \neq 1$ se produce unul din următoarele trei evenimente:

1. Nodul i actualizează $D_i(t)$ după formula:

$$D_i(t) = \min_{j \in N(i)} [d_{ij} + D_j^i(t)]$$

și păstrează $D_j^i(t)$, $j \in N(i)$ neschimbate;

2. Nodul i primește de la $j \in N(i)$ valorile D_j, calculate în j într-un moment de timp anterior, actualizează valorile $D_j^i(t)$, celelalte păstrându-le neschimbate;
3. Nodul i este liber și toate valorile aflate în nod rămân neschimbate.

Fie T^i - mulțimea momentelor de timp în care în nodul i se produc modificări de tip 1 și T_j^i - mulțimea momentelor de timp în care nodul i primește informații de la j de tipul 2.

Presupunem următoarele:

1. Nodurile nu încetează actualizarea propriilor valori și primirea informațiilor de la vecini, adică T^i și T_j^i sunt formate dintr-o infinitate de elemente pentru toți $i \neq 1$ și $j \in N(i)$;
2. Toate valorile inițiale aflate în noduri $D^i(t_0)$ și $D_j^i(t_0)$, $j \in N(i)$ sunt nenegative. În plus, toate valorile transmise nodurilor înainte de t_0, dar primite după t_0 sunt nenegative;
3. Sistemul va șterge în mod obligatoriu informația veche privind distanțele, adică pentru orice moment $t \geq t_0$ există momentul $t'' > t'$, după care orice nod i nu va primi valoarea D_j dacă ea a fost calculată în nodul vecin $j \in N(i)$ până la momentul t'.

Folosim această formalizare pentru a arăta valabilitatea algoritmului distribuit și asincron.

Valabilitatea algoritmului

Următoarea propoziție confirmă faptul că, într-o perioadă de timp limitat valorile $D_i(t)$ devin egale cu distanțele minime corecte.

Propoziție. Există momentul de timp t_m, astfel încât $D_i(t)=D_i$ pentru toți $t \geq t_m$, $i = 1, ..., N$, unde D_i este distanța minimă corectă de la nodul i la nodul destinație (aici notat cu 1).

Demonstrație:

Ideea demonstrației constă în a determina două șiruri ($\underline{D}_i^k$) și ($\overline{D}_i^k$) pentru fiecare nod i, care să îndeplinească inegalitățile:

$$\underline{D}_i^k \leq \underline{D}_i^{k+1} \leq D_i \leq \overline{D}_i^{k+1} \leq \overline{D}_i^k \quad (4)$$

$$\underline{D}_i^k = D_i = \overline{D}_i^k \text{ pentru fiecare } k \text{ suficient de mare} \quad (5)$$

Aceste șiruri se obțin cu ajutorul algoritmului Bellman-Ford, ce folosește condiții inițiale diferite. Apoi se arată că pentru oricare k și i valorile $D_i(t)$ satisfac inegalitățile: $\underline{D}_i^k \leq D_i(t) \leq \overline{D}_i^k$ pentru toți t suficient de mari. Rolul cheie în demonstrație îl joacă *proprietatea de monotonie* a iterațiilor Bellman-Ford care constă în faptul că, dacă pentru două mărimi scalare D_j' și D_j'' se îndeplinește inegalitatea $D_j' \geq D_j''$ pentru toți $j \in N(i)$, atunci inegalitatea se păstrează și după iterație, adică

$$\min_{j \in N(i)} [d_{ij} + D_j'] \geq \min_{j \in N(i)} [d_{ij} + D_j'']$$

Din această inegalitate rezultă că dacă D_i^k sunt șirurile formate în urma iterațiilor Bellman-Ford (1) și (2) cu condițiile inițiale D_i^0, $i = 1, ..., N$ și este adevărată $D_i^1 \geq D_i^0$ pentru orice i, atunci $D_i^{k+1} \geq D_i^k$ pentru toți i și k. Analog, dacă $D_i^1 \leq D_i^0$ pentru orice i, atunci $D_i^{k+1} \leq D_i^k$ pentru toți i și k.

Fie D_i^k, $i = 1, ..., N$ - iterația de ordinul k a algoritmului Bellman-Ford determinat de iterația (1) și (2) având condițiile inițiale $D_i^0 = \infty$, $i=1$ și $D_i^0 = 0$, iar $\underline{D}_i^k$, $i=1, ...,N$ - iterația de ordinul k cu condițiile inițiale $\overline{D}_i^0 = 0$, $i=1, ...,N$.

*Lemă**). Șirurile $(\underline{D}_i^k)$ și $(\overline{D}_i^k)$ determinate mai sus, îndeplinesc relațiile (4) și (5).

Algoritmul lui Bellman-Ford determină drumul de lungime minimă între nodurile rețelei și nodul destinație (fixat). Determinarea drumului de lungime minimă între oricare două noduri se poate face executând în paralel algoritmul lui Bellman-Ford pentru fiecare nod ales ca și destinație.

Pentru aceasta toate nodurile i vor avea următoarele declarații de variabile:

```
var dist_i:tablou[1..N] cu valori în 0..N
        /* inițializat cu 0 pentru indicele i și cu N altfel */
    rutaj:tablou[1..N] cu valori în 1..N
constante  vecini_i = { mulțimea vecinilor lui i }
           {d_ij}_j∈vecini
                    /* d_ij = distanța de la i la vecinul j */
```

Comportamentul nodului i este următorul: vom nota prin *pch(x)* - pachetul ce transportă informația x.

```
Inițial    /* aceasta se execută pentru un număr oarecare
                de noduri, dar cel puțin o dată */
┌■ înc
│   pentru orice j∈vecin_i livrează pch(dist_i) spre j
└■
```

În cazul recepționării *pch(d)* de la j:

```
┌■ Înc
│   Modif ← false;
│ ┌■ pentru k=1,N execută
│ │ ┌■ dacă dist_i[k]>d_ij+d[k];
│ │ │   atunci dist_i[k]←d_ij+d[k];
│ │ │           rutaj_i[k]←j
│ │ │           modif←true;
│ │ └■
│ └■
│ ┌■ dacă modif
│ │   atunci ∀ r ∈ vecin_i : livrează pch(dist_i) spre r
│ └■
└■
```

*) *Pentru a evita încărcarea articolului, nu publicăm demonstrația. Cei interesați pot să obțină articolul complet de la autoare sau redacție.*

Algoritmul prezentat rezolvă prima parte a problemei de rutaj, cea de calculare a tabelelor de rutaj. În cazul rutajului static, tabelele de rutaj se calculează la inițierea sistemului cu prelucrare distribuită și rămân neschimbate pentru întreaga durată de activitate. După calcularea tabelelor de rutaj, funcția algoritmilor de rutaj se reduce la cea de direcționare a pachetelor în rețea, folosind tabelele de rutaj.

Comportamentul nodului i este următorul:

```
                    { Nodul i conține un pachet cu destinație j }
┌■ dacă j=i
│  atunci "livrează pachetul local"
│  altfel "trimite pachetul la nodul rutaj_i[j]"
└■
```

Discuții privind algoritmul

La apariția unor modificări de topologie sau privitor la costul parcurgerii unui canal, pentru actualizarea tabelelor de rutaj, se reexecută algoritmul, recalculând întregul tabel de rutaj. Pentru ca algoritmul să fie eficient, intervalul de timp dintre t_i și t_{i+1} trebuie să fie suficient de mare. Prin urmare modificările survenite prea des în sistemele cu prelucrare distribuită ce folosesc astfel de algoritmi duc la ineficiența acestor sisteme. Aceasta este cauzată de unele neajunsuri ale algoritmului, printre care amintim:

- În unele situații algoritmul necesită un număr mare de iterații pentru calcularea tabelelor de rutaj. Aceasta se întâmplă nu din cauza asincronității algoritmului, ci din cauza alegerii arbitrare a condițiilor inițiale.
- În situații nefavorabile (de exemplu, de topologie) algoritmul poate necesita prea multe livrări de informații.

Bibliografie

1 Bertsekas D., Gallager P., *Data networks*, Editura «Mir», Moscova, 1989

2 Boian F. M., *Programarea distribuită în Internet (metode și aplicații)*, Editura Albastră, Cluj, 1997

3 Bui A., *A bounding problem concerning distribited routing algorithms*. Rev. Roumaine Math. Pures Appl. 39 (1994), 1-15

4 Bui A., *Analyse du functionnement d'un type d'algorithmes*. Rev. Roumaine Math. Pures Appl., 40 (1995), 7-8, 583-592

5 Bui A., *Performance evalution of fault-tolerant routing algorithms: an optimization problem. Optimization*, 29 (1994), 267-280

6 Raynal M., *La communication et le temps dans les reseaux et les systemes repartis*. Editions Eyrolles, Paris, 1991

7 Tel G., *Introduction to Distributed Algorithms*. Cambridge Press, 1994

Viorica Covali este doctorantă cu frecvență la Universitatea Babeș-Bolyai, Cluj; poate fi contactată prin intermediul redacției.

MICROPROCESOARE

Mihai Scorțaru

Microprocesorul poate să rămână o "cutie neagră" pentru mulți utilizatori ai calculatorului...
Știind că printre cititorii fideli ai Ginfo sunt mulți curioși, propunem o aventură pe parcursul a două numere în universul palpitant al microprocesoarelor

Structura internă a microprocesoarelor

Microprocesorul este o *unitate centrală de prelucrare (Central Processing Unit - CPU)* încorporată într-o capsulă de circuit integrat, el fiind coordonatorul tuturor operațiilor efectuate de către calculator. Are rolul de a citi instrucțiunile dintr-o zonă de memorie, de a decodifica și executa comenzile corespunzătoare.

Pentru ca microprocesorul să poată citi codul instrucțiunii care urmează a fi decodificată și executată, el trebuie să genereze o adresă care va fi păstrată în memorie până când din celula selectată pe baza acestei adrese va putea fi citită data cerută. Pentru a putea fi memorată starea liniilor de adresă în timpul operației de citire, microprocesorul are nevoie de o memorie intermediară numită *registru tampon de adrese (Address Buffer - AB).* Informația codificată care a fost citită, trebuie și ea depusă într-un registru intermediar, numit *registru tampon de date (Data Buffer - DB).*

Traseele de cupru cu ajutorul cărora se va genera informația binară care reprezintă o adresă se numesc *magistrală de adrese (Address BUS - ABUS)*, iar cele care vor genera informații corespunzătoare datelor citite sau scrise în memorie poartă denumirea de *magistrală de date (Data BUS - DBUS).*

Pentru a putea efectua diferite calcule aritmetice și logice, microprocesorul are nevoie de o *unitate aritmetică și logică (Arithmetic and Logic Unit - ALU).*

Pentru a putea fi executată o anumită instrucțiune procesorul conține și o *unitate de comandă (Command Unit - CU).* Această componentă programează execuția secvențială în timp a tuturor manevrelor necesare pentru a putea executa instrucțiunea; ea generează semnalele de comandă pentru întregul sistem, dirijează fluxul de date, corelează viteza de lucru a unității centrale cu timpul de acces al memoriei etc. Activitatea sa este "pilotată" de un semnal de ceas a cărui frecvență (deocamdată) este de ordinul Mhz-ilor.

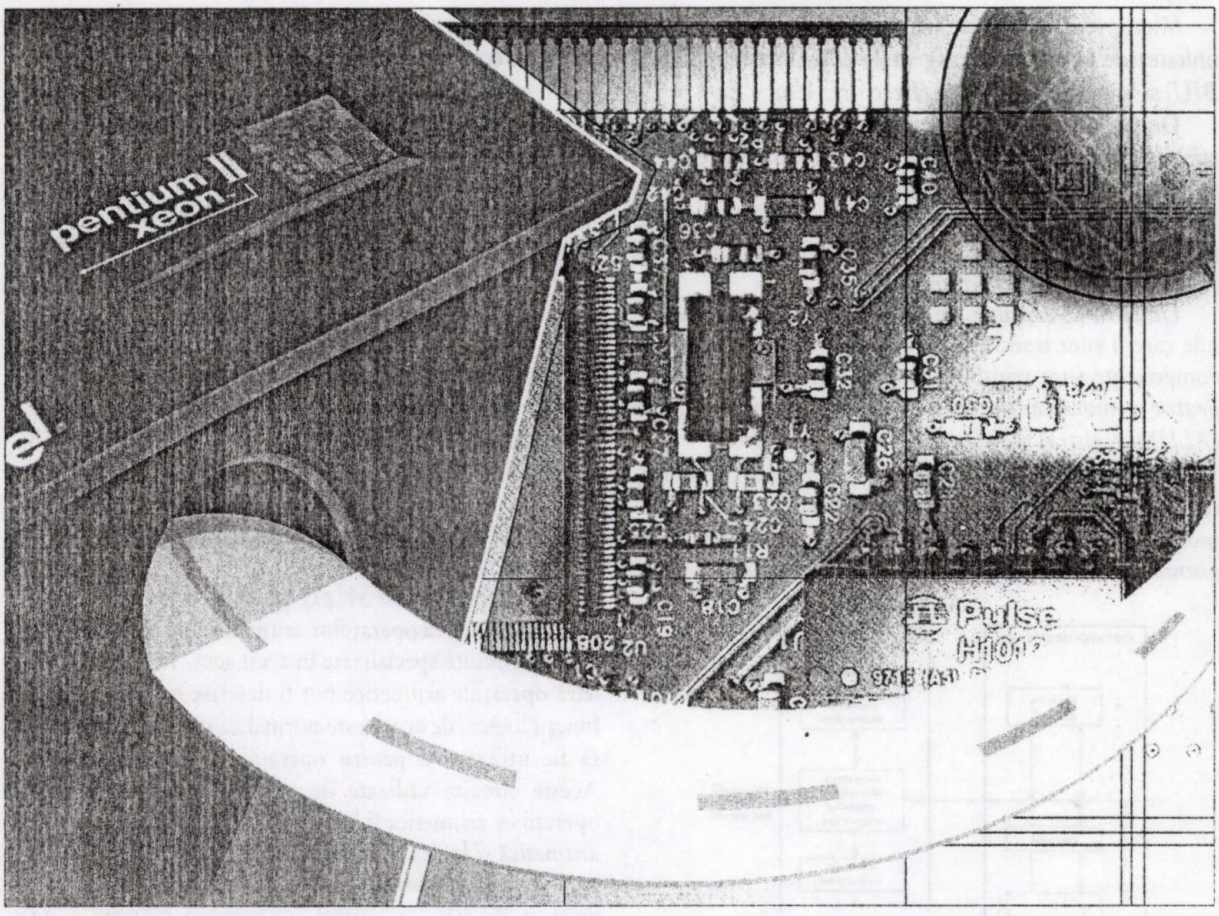
Semnalele prin care microprocesorul dă comenzi de execuție spre memorie sau spre alte componente ale sistemului poartă denumirea de *semnale de comandă.* Semnalele prin care microprocesorul primește informații despre diferite componente ale sistemului se numesc *semnale de stare.* Traseele de cupru pe care circulă aceste semnale formează *magistrala de comenzi (Command BUS - CBUS).*

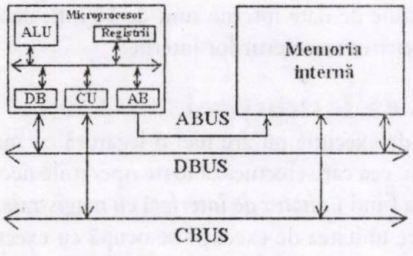
În cazul în care o instrucțiune folosește date care au rezultat în urma execuției instrucțiunii precedente atunci aceste date ar trebui citite din nou din memorie. Pentru a evita acest acces suplimentar la memorie, microprocesorul este prevăzut cu *regiștri*, în care pot fi păstrate temporar date sau adrese de memorie. Microprocesorul mai are nevoie de un registru special în care să poată fi generată și păstrată nealterată adresa de memorie a următoarei instrucțiuni care trebuie executată. Acest registru se numește *instrucțiune program (Instruction Program - IP)* sau *contor de program (Program Counter - PC).* Mai există încă un registru special în care sunt păstrate informații referitoare la natura rezultatului unei operații aritmetice. Biții care compun acest registru se numesc *indicatori de stare,* iar registrul poartă denumirea de *registrul indicatorilor de stare și control.* Vor exista biți corespunzători diferitelor stări ale rezultatului cum ar fi: număr par sau impar, negativ sau pozitiv, nul sau nenul etc.

Pe lângă magistralele externe care sunt folosite pentru "comunicarea" dintre procesor și celelalte componente ale sistemului, microprocesorul este prevăzut și cu o *magistrală internă de date,* necesară pentru efectuarea unor transferuri interne de date.

La magistralele de date și de comenzi pot fi cuplate *circuite de intrare/ieșire (Input/Output - I/O)* care realizează legătura cu exteriorul.

Lățimea unei magistrale reprezintă numărul de trasee de cupru care o compun. Numărul de locații de memorie care pot fi adresate direct de procesor depinde de lățimea magistralei de adrese, de exemplu o magistrală de adrese care conține 20 de trasee permite adresarea a 220 celule de memorie.





Schema simplificată a unui microprocesor și a magistralelor

Pentru a se executa o instrucțiune, au loc următoarele evenimente: microprocesorul depune pe magistrala de date valoarea din registrul IP și depune pe magistrala de comenzi comanda *citește din memorie*; memoria internă preia de pe magistrala de date valoarea *N* care a fost stocată în registrul IP, caută adresa *N*, preia valoarea *I* conținută în această locație, depune pe magistrala de date valoarea *I* și pe magistrala de comenzi comanda *am terminat citirea*; după ce recepționează această comandă, microprocesorul citește de pe magistrala de date valoarea *I* și execută instrucțiunea codificată prin această valoare; valoarea din registrul IP este incrementată cu o unitate (devine *N*+1).

Valoarea registrului IP poate fi modificată de una din instrucțiunile unui program, dispărând astfel monotonia cauzată de incrementarea cu o unitate. O astfel de instrucțiune ar putea fi *scade valoarea k din valoarea registrului IP*. Efectul acestei instrucțiuni este întoarcerea cu *k* instrucțiuni înainte și repetarea lor în alt context (valorile datelor vor fi altele).

Toate activitățile calculatorului sunt coordonate de către pulsurile unui oscilator numit ***ceas intern***. După cum am văzut, execuția unei instrucțiuni constă din execuția mai multor etape parțiale. Fiecare astfel de etapă parțială este efectuată în intervalul dintre două pulsuri de ceas, astfel că intervalul de timp necesar execuției unei instrucțiuni poate fi măsurat cu ajutorul numărului de pulsuri de ceas.

Structura unui microprocesor Intel

În continuare ne vom ocupa în principal de microprocesoarele 8086 și 8088, acestea fiind denumite de cărțile de specialitate ***microprocesoare convenționale***. Ori de câte ori vom face referire la un alt microprocesor îi vom indica numele.

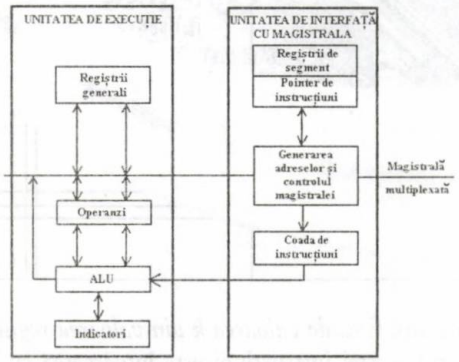
Microprocesoarele 8086 și 8088 au o structură pe 16 biți, deosebirea esențială dintre ele constând în faptul că 8088 este proiectat cu o magistrală de date externă de 8 biți, în timp ce 8086 dispune de una de 16 biți. Aceasta înseamnă că pot fi transferați pe magistrala de date externă 8, respectiv 16 biți deodată. Din motive comerciale, Intel a păstrat compatibilitatea cu microprocesoarele apărute anterior, ceea ce înseamnă că orice program care funcționează cu un anumit microprocesor va funcționa și cu orice alt microprocesor apărut ulterior.

Microprocesorul Intel are două componente majore: *unitatea de interfață cu magistrala (Bus Interface Unit - BIU)* și *unitatea de execuție (Executive Unit - EU).*

Unitatea de interfață cu magistrala transmite procesorului instrucțiuni și operanzi și rezultatele unor prelucrări spre memoria internă sau dispozitivele de intrare/ieșire. Principalele componente ale sale sunt *regiștrii de segment, registrul pointer de instrucțiuni, dispozitivul de adresare și coada fluxului de instrucțiuni.*

Unitatea de execuție are rolul de a executa instrucțiunile care îi sunt transmise de către BIU. Principalele sale componente sunt *regiștrii generali, registrul de manevră pentru memorarea operanzilor, unitatea aritmetico-logică (ALU) și registrul indicatorilor de stare și control.*

Magistralele cu ajutorul cărora se realizează comunicarea dintre microprocesor și celelalte componente ale sistemului de calcul sunt: *magistrala de date* (pe 16 biți), *magistrala de adresă* (pe 20 de biți) și *magistrala de comenzi.*



Structura unui microprocesor Intel

Sistemul de întreruperi

Microprocesorul mai dispune de două linii pentru semnalele de întrerupere emise de dispozitivele externe: linia *INTR (Interrupt Request)* și linia *NMI (Non-Maskable Interrupt).* Pe linia INTR sosesc întreruperi în urma cărora unitatea centrală, în funcție de valoarea *indicatorului de validare a întreruperilor (IF)*, execută diferite operații. Dacă IF este șters (are valoarea 0) înseamnă că întreruperile semnalate pe linia INTR sunt mascate sau dezactivate, ca urmare unitatea centrală le ignoră și trece la instrucțiunea următoare. Dacă IF este poziționat (are valoarea 1) înseamnă că întreruperile de pe linia INTR sunt activate și unitatea centrală oprește execuția normală a instrucțiunilor și predă controlul rutinei de tratare a întreruperilor. NMI este o linie prin care sosesc întreruperi care anunță evenimente critice. Aceste întreruperi nu pot fi dezactivate și unitatea centrală le va lua în considerare de fiecare dată când acestea sosesc.

Sistemul de întreruperi este simplu și ușor de manevrat. Pot fi utilizate cel mult 256 de tipuri de întrerupere, fiecărui tip fiindu-i atribuit un cod specific. Întreruperile pot fi clasificate în întreruperi *externe* și întreruperi *interne*. Cele externe sunt numite și *întreruperi hardware*, iar cele interne mai poartă denumirea de *întreruperi software*. Diferența esențială dintre cele două tipuri de întreruperi este că cele externe sunt cauzate de dispozitivele externe, iar cele interne sunt generate prin program, folosindu-se anumite instrucțiuni.

Tabela vectorilor de întrerupere

Tabela vectorilor de întrerupere permite asocierea dintre tipul întreruperii (reprezentat de un număr între 0 și 255) și rutina de tratare a întreruperii. Fiecare intrare a tabelei ocupă 4 bytes, ea conținând adresa rutinei de tratare a întreruperii. Tabela ocupă prima zonă a memoriei interne și poate avea până la 256 de intrări.

La apariția unei întreruperi este suspendată execuția instrucțiunilor programului curent și sunt salvate pe stivă registrul cu indicatorii de stare și control și adresa instrucțiunii următoare; apoi este activată rutina de tratare a întreruperii.

Unitatea aritmetică și logică

Pentru efectuarea operațiilor aritmetice procesorul are nevoie de circuite specializate în acest sens. În aritmetica binară operațiile aritmetice pot fi descrise cu ajutorul unor funcții logice, de aceea este normal ca circuitele respective să fie utilizate și pentru operațiile logice propriu-zise. Aceste circuite utilizate de procesor pentru efectuarea operațiilor aritmetice și logice poartă denumirea de *unitate aritmetică și logică.* ALU face parte din unitatea de execuție a microprocesorului, ea având și rolul de a gestiona indicatorii de stare și control ai microprocesorului și de a manevra regiștrii generali și operanzii instrucțiunilor. Regiștrii și căile de date interne sunt de 16 biți, deoarece se dorește mărirea transferurilor interne.

Coada de instrucțiuni

Unitatea de execuție nu are nici o legătură cu magistrala sistemului, cea care efectuează toate operațiile necesare cu magistrala fiind *unitatea de interfață cu magistrala.* În timpul în care unitatea de execuție se ocupă cu execuția instrucțiunilor, unitatea de interfață cu magistrala încarcă instrucțiunile într-o memorie specială de tip RAM (Random Acces Memory) numită *coada fluxului de instrucțiuni.* Această memorie are o dimensiune de 4 bytes în cazul microprocesoarelor 8088 și 6 bytes în cazul 8086. Unitatea de execuție preia câte o instrucțiune din coada fluxului de instrucțiuni; apoi o execută.

În majoritatea cazurilor coada fluxului de instrucțiuni conține cel puțin un byte al fluxului de instrucțiuni și unitatea de execuție nu mai trebuie să aștepte încărcarea instrucțiunii care urmează a fi executată. La un moment dat în coada fluxului de instrucțiuni sunt memorate instrucțiunile care se află în locațiile de memorie imediat următoare locației de memorie în care se află instrucțiunea care se execută la acel moment. Dacă este executată o instrucțiune care transferă controlul unei locații de memorie, coada fluxului de instrucțiuni este ștearsă, se încarcă instrucțiunea de la noua adresă și unitatea de interfață cu magistrala în-

cepe reîncărcarea cozii fluxului de instrucțiuni cu instrucțiunile următoare.

Dispozitivul de adresare

Microprocesorul trebuie să poată adresa atât instrucțiunile cît și datele de care instrucțiunile au nevoie atunci când sunt executate. Adresa locației de memorie din care se va încărca o instrucțiune este suma dintre o adresă de segment și un deplasament. Pentru adresarea datelor, microprocesorul oferă câteva modalități de adresare.

Segmentarea memoriei

Microprocesoarele 8086 și 8088 pot adresa 1 MB de memorie care sunt grupați în mai multe segmente. Un *segment* de memorie este o unitate logică de memorie care poate avea cel mult 64 KB. Fiecare segment de memorie este alcătuit din mai multe locații de memorie consecutive și poate fi considerat ca o unitate care poate fi adresată independent. Fiecare segment este caracterizat printr-o adresă de bază care reprezintă adresa locației de început a segmentului și este exprimată în număr de blocuri de 16 bytes, numite *paragrafe*. Ca urmare, toate segmentele încep de la o adresă care este un multiplu de 16. Două segmente se numesc *adiacente* dacă adresa de început a unuia dintre ele este cu 1 mai mare decât adresa de sfârșit a celuilalt. Două segmente sunt parțial suprapuse dacă adresa de început a unuia se află între adresa de început și cea de sfârșit a celuilalt. Două segmente sunt suprapuse complet dacă adresele lor de început coincid. În toate celelalte cazuri segmentele se numesc *disjuncte*.

În funcție de necesități o anumită aplicație (program) utilizează segmentele în mod diferit. Segmentele care pot fi adresate în mod curent oferă 64 KB pentru cod, 64 KB pentru stivă și 64 KB pentru date. Pentru a utiliza un spațiu de lucru mai mare, trebuie gestionate segmentele de memorie în funcție de necesități.

Adrese fizice și adrese logice

Adresa fizică este o valoare reprezentată pe 20 de biți care identifică în mod unic fiecare locație de memorie aflată în spațiul de adresare. Ca urmare această adresă poate fi un număr cuprins între 0 și 220-1.

Pentru a nu depinde de locul în care este păstrat codul în memorie, programele folosesc adresele logice în locul celor fizice. *Adresa logică* este dată de o valoare de bază de segment și o valoare de deplasament. Valoarea de bază a segmentului indică adresa primului byte al segmentului care conține locația și este exprimată în paragrafe. Deplasamentul reprezintă numărul de octeți dintre locația respectivă și începutul segmentului. Atât adresa de bază cât și deplasamentul sunt reprezentate pe 16 biți.

Este posibil ca mai multe adrese logice să localizeze o aceeași locație de memorie atunci când aceasta face parte din segmente diferite. De exemplu adresele logice F000:FFF0 și FFFF:0000 indică aceeași locație de memorie și anume cea care are adresa fizică FFFF0. Generarea

Microprocesoare Intel

Procesoarele Intel sunt grupate în mai multe generații în funcție de anumite caracteristici.

Prima generație de microprocesoare realizate de Intel a debutat în 1971 cu microprocesorul 4004 pe 4 biți, acesta fiind repede îmbunătățit o dată cu apariția microprocesorului 8008.

În 1974 Intel realizează a *doua generație* de microprocesoare pe 8 biți al cărui reprezentant de bază este 8080, acesta având o deosebită importanță pentru industria microcalculatoarelor.

În 1978 este realizată *a treia generație* de microprocesoare, trecându-se la procesoarele pe 16 biți. Microprocesorul 8088 a fost folosit în 1981 la realizarea calculatorului IBM PC, cel care a dat startul în revoluția calculatoarelor personale. Această familie de microprocesoare, din care face parte și 8086, a fost denumită familia 80x86.

În 1982 apare microprocesorul 80186 care era identic cu 8086 din punct de vedere arhitectural, dar conținea câteva dispozitive sistem încorporate în același circuit.

Tot în 1982 apare 80286 cu o arhitectură extinsă față de 8086; începând cu acest procesor sunt introduse mecanismele de memorie virtuală, multitasking și protecție.

În 1985 este lansat 80386 care face saltul la microprocesoarele pe 32 de biți.

În 1989 apare 80486 care respectă arhitectura microprocesorului 80386, dar înglobează și coprocesorul matematic 80387.

În 1993 firma Intel începe livrarea următorului microprocesor din familia 80x86 rupând tradiția potrivit căreia ar fi trebuit să se numească 80586 și denumindu-și noul microprocesor Pentium.

În ultimii ani au apărut mai multe alte microprocesoare Intel, cele mai importante fiind Pentium PRO, Pentium MMX și Pentium II.

Istoria microprocesoarelor din familia 80x86 este strâns legată de cea a coprocesoarelor matematice, fiecărui microprocesor (până la apariția lui 80486) fiindu-i asociat câte un coprocesor matematic necesar pentru îmbunătățirea performanțelor sistemului de calcul. Pentru microprocesoarele 8086, 80286 și 80386 au fost folosite microprocesoarele 8087, 80287 și 80387. Începând cu 80486 coprocesorul a fost încorporat în microprocesor.

adresei fizice cu ajutorul adresei logice se realizează înmulțind cu 16 valoarea bazei segmentului (shiftare spre stânga cu 4 poziții) și adăugând deplasamentul.

Mecanisme de adresare

Calculele de adresă sunt efectuate de *unitatea de interfață cu magistrala,* cu ajutorul unei unități de calcul, specifice, numită *ADR* și a unor regiștri speciali fiecare având o capacitate de 16 biți.

Primele generații de calculatoare aveau o memorie de 1 MB. Primele 10 blocuri de memorie (640 KB) formează memoria RAM, următoarele 5 fiind folosite pentru memoria ecran și reținerea conținutului memoriei ROM în timpul funcționării sistemului. Pentru a putea codifica în mod unic fiecare din cele 220 de locații de memorie diferite, o adresă trebuie să fie reprezentată pe 20 de biți. Magistrala de adrese are capacitatea necesară pentru a transmite cei 20 de biți deodată, însă nu se poate spune același lucru despre magistrala de date care are o capacitate de 16 biți. Prin convenție adresa de început a fiecărui segment este un multiplu de 16, iar un segment poate avea cel mult 64 KB. Datorită acestui fapt cei mai nesemnificativi 4 bytes ai adresei de început a segmentului vor fi 0 (în baza 2 un multiplu de 16 are ultimele 4 cifre întotdeauna 0). Din acest motiv acești 4 biți nu mai trebuie memorați și ca urmare regiștrii de segment vor conține numai restul celor 16 biți. Pentru a specifica deplasamentul (*offset*) sunt folosiți tot 16 biți.

O *adresă virtuală* este compusă din două numere reprezentate fiecare pe 16 biți, unul reprezentând conținutul registrului de segment, iar al doilea deplasamentul în cadrul segmentului. Adresa fizică este calculată de către BIU înmulțind cu 16 primul număr și adunând la rezultat cel de-al doilea număr.

Mecanismul descris este tipic pentru procesorul 8086 și poartă denumirea de *mod real (Real Adress Mode).* La microprocesoarele 80286 apare încă un mod de adresare, denumit *mod protejat (Protected Virtual Address Mode),* iar o dată cu apariția procesorului 80386 sunt introduse *modul paginat* și *modul virtual 8086.* Aceste noi modalități de adresare au fost introduse pentru a permite adresarea de către un calculator IBM-PC a mai mult de 1 MB de memorie.

Implementarea stivei

Stiva este o zonă de memorie folosită pentru a stoca datele temporare. Deosebirea esențială dintre stive și alte segmente de memorie este faptul că datele de pe stivă sunt stocate începând de la adresele mai mari de memorie și continuând spre adrese mai mici. La început stiva este un segment de memorie neinițializat care are o dimensiune fixă. În timp ce sunt adăugate informații în stivă dimensiunea stivei "crește" de la adresele mari spre adresele mai mici, iar când sunt eliminate informații, stiva se "micșorează" în sens invers.

Numărul de stive este limitat numai de spațiul de memorie disponibil în sistem, iar dimensiunea unei stive nu poate depăși 64 KB. Dacă se extinde stiva peste această dimensiune, informațiile adăugate vor înlocui informațiile de la începutul stivei. La un moment dat nu poate fi adresată direct decât o stivă, ea purtând denumirea de ***stivă curentă.***

Regiștri

Regiștrii microprocesoarelor 8086 și 8088 sunt în număr de 14, au fiecare câte 16 biți și pot fi împărțiți în următoarele categorii: *regiștrii generali, regiștrii de segment, registrul pointerului de instrucțiuni și registrul indicatorului de stare și control.*

Regiștrii generali sunt în număr de 8 și pot fi clasificați în regiștri de date (AX, BX, CX și DX), regiștri de pointer (SP și BP) și regiștri de index (SI și DI). O caracteristică importantă a regiștrilor de date este aceea că "jumătățile" lor pot fi accesate separat, deci ei pot fi folosiți fie ca regiștri de 16 biți, fie ca doi regiștri de câte 8 biți fiecare. Cele două jumătăți ale registrului AX pot fi referite ca regiștri AL, respectiv AH. În același mod putem folosi regiștrii BL, BH, CL, CH, DL și DH. Regiștrii de pointer și de index pot fi folosiți numai ca regiștri de 16 biți.

Regiștrii de segment, 4 la număr, sunt denumiți CS, SS, DS și ES. Registrul CS indică segmentul de cod curent (adresa de bază a segmentului de memorie care conține codul programului). Registrul SS indică segmentul de stivă curent, operațiile cu stiva fiind efectuate în locațiile de memorie corespunzătoare acestui segment. Registrul DS indică segmentul de date curent, segment ce conține, de obicei, datele și variabilele programului. În sfârșit, registrul ES, indică extrasegmentul curent, folosit de obicei tot pentru date. Acești regiștri pot fi modificați direct prin program.

Registrul pointerului de instrucțiuni (IP) este un registru actualizat de BIU și conține deplasamentul locației de memorie în care se află instrucțiunea următoare față de începutul segmentului de cod curent. Acest registru nu poate fi modificat direct prin program.

Registrul indicatorilor de stare și control este un registru ai cărui biți au fiecare câte o semnificație aparte. Ei indică starea rezultatului operației pe care o execută programul la un moment dat sau controlează unele operații ale microprocesorului. Registrul conține 6 biți de stare și 3 biți de control, restul biților fiind rezervați, ei neavînd o semnificație precizată.

Microprocesorul 80386 utilizează aceiași regiștri ca și predecesorii săi din familia 8086, dar toți regiștrii (cu excepția celor de segment) pot fi extinse până la 32 de biți. Denumirea regiștrilor extinși începe cu litera *E* și ei se numesc EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI și EIP). Au fost adăugate și doi regiștri de segment suplimentari FS și GS.

Indicatori

Cei 16 biți ai registrului indicatorilor de stare și control poartă denumirea de *indicatori* (denumire sugerată și de numele registrului) sau *flaguri*. Cuvântul englezesc *flag* este tradus în limba română prin *steag*. Denumirea sugerează faptul că un anumit steguleț va fi ridicat în cazul în care o condiție este îndeplinită. Mai multe instrucțiuni permit modificarea execuției unui program în funcție de cei șase indicatori de stare.

Indicatorul de transport (Carry - CF) indică faptul că cel mai semnificativ bit al rezultatului operației care s-a efectuat a fost "transportat" în exterior. De obicei este utilizat atunci când se efectuează operații matematice cu numere a căror reprezentare este pe mai mult de un octet. Indicatorul mai poate fi modificat și în cazul operațiilor de deplasare și rotire.

Indicatorul de paritate (Parity - PF) este poziționat dacă rezultatul conține în reprezentare un număr par de biți cu valoarea 1. De obicei este folosit pentru verificarea transmisiilor de date.

Indicatorul de transport auxiliar (Auxiliary Carry - AF) este poziționat dacă a avut loc un transport de la un nivel inferior la un nivel superior al semioctetului rezultatului. Este folosit în operațiile cu numere reale.

Indicatorul de zero (Zero - ZF) este poziționat dacă rezultatul operației este 0.

Indicatorul de semn (Sign - SF) este poziționat dacă cel mai semnificativ bit al rezultatului este 1.

Indicatorul de depășire (Overflow - OF) este poziționat dacă are loc o depășire aritmetică, adică dimensiunea rezultatului depășește capacitatea locației de destinație. Acești biți indicatori de stare sunt poziționați de către unitatea de execuție. Indicatorii de control permit controlarea unor operații ale microprocesorului și ei pot fi modificați prin program.

Indicatorul de depanare (Trap/Trace - TF) dacă este poziționat, indică faptul că programul va fi executat pas cu pas. Unitatea centrală va genera o întrerupere după fiecare instrucțiune, existând posibilitatea urmăririi execuției fiecărei instrucțiuni.

Indicatorul de întrerupere (Interrupt - IF) indică unității centrale, în cazul în care este poziționat, că trebuie să "reacționeze" la întreruperile externe mascabile. Dacă acest indicator nu este poziționat aceste întreruperi vor fi ignorate. Nu are efect asupra întreruperilor externe nemascabile sau asupra întreruperilor interne.

Indicatorul de direcție (Direction - DF) în cazul în care este poziționat, indică faptul că șirurile vor fi prelucrate de la adresele mari spre adresele mici (de la dreapta la stânga). În caz contrar șirurile vor fi prelucrate de la stânga la dreapta.

Începând cu microprocesorul 80286 au fost adăugate încă două indicatoare pentru a acorda mai multe facilități programatorului.

Indicatorul protecție intrare/ieșire (Input/Output Privilege level flag - IOP) indică nivelul de protecție pentru intrare și ieșire. Trebuie remarcat că pentru acest flag sunt folosiți 2 biți.

Indicatorul imbricare (Nested flag - N) controlează înlănțuirea proceselor întrerupte în modul protejat.

Începând cu microprocesorul 80386 registrul indicatorilor de stare și control este și el extins la 32 de biți, fiind adăugați încă doi indicatori.

Indicatorul reluare (Restart flag - R) determină, dacă este poziționat, dezactivarea temporară a excepțiilor de depanare.

Indicatorul mod virtual (Virtual 8086 mode flag - V) indică, în cazul poziționării sale, modul de execuție virtual 8086. În caz contrar modul de execuție va fi modul protejat.

Tabelul 1 indică biții folosiți pentru fiecare flag în parte:

O dată cu apariția microprocesorului 80486 au apărut încă 7 indicatori de stare și control: indicatorul AC permite generarea unui semnal de eroare în cazul în care apar date incorect aliniate; indicatorul PG arată dacă paginarea este activă sau nu; indicatorul CE este folosit pentru conectarea memoriei cache; indicatorul WT este folosit pentru validarea și invalidarea scrierii în memoria cache; indicatorul WE controlează dacă la coprocesor apare o excepție datorată vectorului de întrerupere 2h sau 16h (numerele terminate în h indică faptul că ele sunt scrise în sistem hexazecimal - baza 16; numărul 16h poate fi scris ca 22 dacă se folosește sistemul zecimal); indicatorul TS este setat în cazul în care este lansat un task; indicatorul EM arată dacă instrucțiunile ESC sunt preluate sau transmise mai departe la coprocesor.

Promisiuni

În numărul următor al "Gazetei de Informatică" veți putea afla mai multe despre instrucțiunile microprocesorului, despre reprezentarea lor în cod mașină și despre instrucțiunile limbajului de asamblare.

Mihai Scorțaru este student la Universitatea Tehnică din Cluj. Poate fi contactat prin e-mail la adresa: scort@licj.soroscj.ro

Tabelul 1.

Bit	Flag	Bit	Flag	Bit	Flag	Bit	Flag	Bit	Flag	Bit	Flag	Bit	Flag	Bit	Flag
0	CF	4	AF	8	TF	12	IOP	16	R	20	-	24	-	28	-
1	-	5	-	9	IF	13	IOP	17	V	21	-	25	-	29	-
2	PF	6	ZF	10	DF	14	N	18	-	22	-	26	-	30	-
3	-	7	SF	11	OF	15	-	19	-	23	-	27	-	31	-

Programarea PLĂCII de SUNET

Adrian Monea

După ce în numărul trecut am învățat despre programarea chip-ului FM al plăcii Sound Blaster, în acest număr vom învăța despre programarea chip-ului DSP. De asemenea va fi prezentat și un unit Pascal pentru lucrul cu DSP-ul.

Porturile chip-ului DSP al plăcii SoundBlaster

Chip-ul DSP (*Digital Sound Processor*) este programat prin intermediul a patru porturi determinate de setarea adresei de bază a plăcii de sunet:

RESET:	$2x6
READ DATA:	$2xA
WRITE COMMAND/DATA output	
WRITE BUFFER STATUS input	$2xC
DATA AVAILABLE	$2xE

Unde:

x = 1 ⇒ adresa de bază a plăcii este setată la $210
x = 2 ⇒ adresa de bază a plăcii este setată la $220
...
x = 6 ⇒ adresa de bază a plăcii este setată la $260

Resetarea DSP-ului

Înainte de a fi programat, DSP-ul trebuie resetat. Acest lucru se face conform următoarei proceduri:

1) Se scrie valoarea 1 în portul de RESET ($2x6)
2) Se așteaptă 3 microsecunde
3) Se scrie valoarea 0 în portul de RESET ($2x6)
4) Se citește octetul de la portul de date disponibile ($2xE) până când bitul 7 este 1
5) Se citește octetul de la portul de citire date ($2xA) până când valoarea acestuia este $AA.

DSP-ul are nevoie în mod normal de aproximativ 100 de microsecunde pentru a se reseta. Dacă nu se resetează într-un timp rezonabil (să zicem 200 de microsecunde) atunci a apărut o eroare (este foarte probabil să se fi folosit o adresă I/O incorectă).

Scrierea în DSP

O valoare poate fi scrisă în DSP astfel:

1) Se citește portul WRITE BUFFER STATUS ($2xC) până când bitul 7 este 0
2) Se scrie valoarea în portul WRITE COMMAND/DATA ($2xC)

Citirea din DSP:

O valoare poate fi citită din DSP astfel:

1) Se citește valoarea din portul DATA AVAILABLE ($2xE) până când bitul 7 este 1
2) Se citește valoarea din portul READ DATA ($2xA)

Activarea difuzoarelor și controlul DMA-ului

Difuzoarele și DMA-ul se controlează scriind unul din următorii octeți în DSP:

Valoare	*Descriere*
$D0	Oprește DMA-ul
$D1	Activează difuzoarele
$D3	Dezactivează difuzoarele
$D4	Repornește DMA-ul

Despre DMA vom discuta în rândurile ce urmează. Comenzile DMA de mai jos pot fi folosite pentru a opri melodia în timpul redării acesteia.

Scrierea în DAC

DAC-ul (*Digital to Analog Converter*) este partea plăcii de sunet responsabilă cu convertirea unui număr (de la 0 la 255) într-un nivel de sunet. Pentru a genera o undă de sunet pătrată la volum maxim (de exemplu) se pot scrie alternativ valori de 0 și 255 în DAC.

Pentru programarea DAC-ului în mod direct programul principal trebuie să seteze valoarea dorită în DAC. În mod direct DAC-ul nu poate fi programat decât pe 8 biți. Pentru a seta nivelul DAC-ului trebuie scrisă valoarea $10 în DSP urmată de numărul dorit (0-255). Evident, dacă difuzoarele nu au fost activate, nu se va auzi nici un sunet. În programarea în mod direct programul este responsabil de sincronizarea eșantioanelor digitale reproduse de placă. În mod tipic un program care folosește DAC-ul în mod direct reprogramează întreruperea de timer a calculatorului pentru a sincroniza eșantioanele necesare redării unei melodii.

DAC-ul poate fi de asemenea programat să accepte valori trimise lui prin controlerul DMA (*Direct Memory Access*). Programarea DMA-ului este complexă și de aceea

în acest articol nu poate fi tratată exhaustiv. Vom aborda totuși câteva aspecte legate de DMA. Cel mai important lucru care nu trebuie uitat în legătură cu controlerul DMA este că acesta nu poate transfera date care se întind pe mai multe pagini de memorie. Dacă datele ce se doresc a fi transferate depășesc în dimensiune o pagină de memorie ele vor trebui transferate în memorie pe bucăți, câte o pagină pe transfer.

Setarea frecvenței de redare pentru transferul DMA este făcută scriind valoarea $40 în DSP urmată de TIME_CONSTANT, unde TIME_CONSTANT=256-1000000/frecvență.

Tipuri de transferuri DMA

Valoare tip_DMA	Descriere	Domeniul de frecvențe
$14	8 biți	4 KHz - 23 KHz
$74	4 biți ADPCM	4 KHz - 12 KHz
$75	4 biți ADPCM cu octet de referință	4 KHz - 12 KHz
$76	2.6 biți ADPCM	4 KHz - 13 KHz
$77	2.6 biți ADPCM cu octet de referință	4 KHz - 13 KHz
$16	2 biți ADPCM	4 KHz - 11 KHz
$17	2 biți ADPCM cu octet de referință	4 KHz - 11 KHz

ADPCM este un acronim pentru *Adaptive Pulse Code Modulation*, o tehnică pentru compresia de sunet în care se stochează diferența dintre eșantioanele consecutive și nu valorile propriu-zise ale acestora. În modurile cu octeți de referință primul octet este de fapt valoarea de start. Existența modurilor cu și fără octeți de referință înseamnă că putem reda blocuri consecutive fără a avea nevoie de un octet de referință la începutul fiecăruia.

Modul de efectuare al unui transfer DMA este următorul:

1) Se încarcă datele de sunet în memorie
2) Se pregătește controlerul DMA pentru transfer
3) Se setează constanta DSP TIME_CONSTANT la rata de eșantionare
4) Se scrie valoarea pentru DMA_TYPE_VALUE în DSP
5) Se scrie valoarea DATA_LENGTH în DSP (2 octeți, cel mai nesemnificativ fiind primul), unde:
   DATA_LENGTH = număr de octeți de trimis - 1

Trebuie reținut că chip-ul DMA trebuie programat înaintea DSP-ului.

Citirea din ADC

Citirea de eșantioane din ADC (*Analog to Digital Converter*) poate fi făcută de asemenea în mod direct sau prin DMA.

Pentru a citi un eșantion în mod direct scrieți valoarea $20 în DSP și apoi citiți valoarea din DSP.

Pentru a pregăti DSP-ul pentru un transfer DMA trebuie urmați următorii pași:

1) Alocați memorie pentru eșantion
2) Pregătiți controlerul DMA pentru transfer
3) Setați constanta DSP TIME_CONSTANT la frecvența de eșantionare
4) Scrieți valoarea $24 în DSP
5) Scrieți DATA_LENGTH în DSP (2 octeți, cel mai nesemnificativ primul), unde DATA_LENGTH= numărul de octeți de citit - 1

Citirile DMA pot fi făcute doar pe 8 biți, modurile cu compresie sunt făcute prin soft și stocate în fișiere cu extensia VOC.

Programarea controlerului DMA

În cele de mai jos vor fi prezentați pașii necesari pentru programarea canalului 1 DMA pentru DSP în modul real:

1) Se calculează adresa de 20 de biți a tamponului de memorie folosit, unde
   *Adresa_de_bază = Segment*16 + Offset,*
   de exemplu $1234:$5678 = $179B8
2) Se scrie valoarea $05 în portul $0A
3) Se scrie valoarea $00 în portul $0C
4) Se scrie valoarea $49 în portul $0B (pentru redare) sau $45 în portul $0B (pentru înregistrare)
5) Se scrie octetul cel mai nesemnificativ (biții 0 - 7) al adresei de memorie de 20 de biți în portul $02
6) Se scrie octetul cel mai semnificativ (biții 8 - 15) al adresei de memorie de 20 de biți în portul $02
7) Se scrie *pagina* (biții 16 - 19) adresei de memorie de 20 de biți în portul $83
8) Se scrie octetul cel mai nesemnificativ al variabilei DATA_LENGTH în portul $03
9) Se scrie octetul cel mai semnificativ al variabilei DATA_LENGTH în portul $03
10) Se scrie valoarea $01 în portul $0A

Terminarea transferului DMA

Când un transfer DMA este complet, este generată o întrerupere. Numărul propriu zis al întreruperii depinde de setarea numărului IRQ pentru placa de sunet:

Număr IRQ	Întrerupere
2	$0A
3	$0B
5	$0D
7	$0F

Pentru a servi una din aceste întreruperi trebuie făcute următoarele:

1) Se recunoaște întreruperea DSP citind portul DATA AVAILABLE ($2xE) o dată
2) Dacă există mai multe blocuri de transfer atunci ele sunt pregătite
3) Se scrie valoarea $20 (EOI) în portul controlerului de întreruperi $20

La fel ca și cu orice altă întrerupere hardware, trebuie refăcută starea sistemului (regiștrii etc) exact la fel cum era înainte de apelarea întreruperii.

Un unit Pascal pentru DSP

În continuare vom prezenta un unit Pascal pentru lucrul cu DSP-ul în modul real, unit scris de ***Mark Feldman***, autorul *PCGPE*.

Listing DSP.PAS

```
Unit DSP;
Interface
{ ResetDSP returneaza True daca resetarea a }
{ avut loc cu succes  base trebuie sa fie 1 }
{ pentru adresa de baza $210,2 pentru $220... }
function ResetDSP(base:Word):Boolean;
    { WriteDAC seteaza nivelul de iesire al }
                                { difuzoarelor }
procedure WriteDAC(level:Byte);
    { ReadDAC citeste nivelul de intrare al }
                                { microfonului }
function ReadDAC:Byte;
 { SpeakerOn conecteaza DAC-ul la difuzoare }
function SpeakerOn:Byte;
    { SpeakerOff deconecteaza DAC-ul de la }
    { difuzoare; nu afecteaza restul       }
    { operatiilor DAC-ului                 }
function SpeakerOff: byte;
    { Functii pentru controlul redarii DMA }
procedure DMAStop;procedure DMAContinue;
{ Playback reda un esantion de o marime data }
{ la o frecventa data folosind canalul 1 DMA }
{ Esantionul nu trebuie sa treaca de o pagina }
procedure Playback(sound:Pointer; size:Word;
                                frequency:Word);
Implementation
Uses Crt;
var DSP_RESET:Word;
    DSP_READ_DATA:Word;
    DSP_WRITE_DATA:Word;
    DSP_WRITE_STATUS:Word;
    DSP_DATA_AVAIL:Word;

function ResetDSP(base:Word):Boolean;
begin
  base:=base*$10;
          { Calculeaza adresele porturilor }
  DSP_RESET:=base+$206;
  DSP_READ_DATA:=base+$20A;
  DSP_WRITE_DATA:=base+$20C;
  DSP_WRITE_STATUS:=base+$20C;
  DSP_DATA_AVAIL:=base+$20E;
 { Reseteaza DSP-ul si produce niste pauze }
 { lungi pentru siguranta                  }
  Port[DSP_RESET]:=1; Delay(10);
  Port[DSP_RESET]:=0; Delay(10);
  ResetDSP:=(Port[DSP_DATA_AVAIL] and
      $80=$80) and (Port[DSP_READ_DATA]=$AA)
end;

procedure WriteDSP(value:Byte);
begin
  while Port[DSP_WRITE_STATUS] and $80<>0 do;
  Port[DSP_WRITE_DATA]:=value
end;

function ReadDSP:Byte;
begin
  while Port[DSP_DATA_AVAIL] and $80=0 do;
  ReadDSP:=Port[DSP_READ_DATA]
end;

procedure WriteDAC(level:Byte);
begin WriteDSP($10); WriteDSP(level) end;

function ReadDAC:Byte;
begin WriteDSP($20); ReadDAC:=ReadDSP end;

function SpeakerOn:Byte;
begin WriteDSP($D1) end;

function SpeakerOff:Byte;
begin WriteDSP($D3) end;

procedure DMAContinue;
begin WriteDSP($D4) end;

procedure DMAStop;
begin WriteDSP($D0) end;

procedure Playback(sound:Pointer;size:Word;
                                frequency:Word);
var time_constant:Word;
    page,offset:Word;
begin
  SpeakerOn;
  size:=size-1;{ Pregateste controlerul DMA }
  offset:=Seg(sound^) shl 4+Ofs(sound^);
  page:=(Seg(sound^)+Ofs(sound^) shr 4)shr 12;
  Port[$0A]:=5; Port[$0C]:=0; Port[$0B]:=$49;
  Port[$02]:=Lo(offset);
  Port[$02]:=Hi(offset); Port[$83]:=page;
  Port[$03]:=Lo(size); Port[$03]:=Hi(size);
  Port[$0A]:=1;
            { Seteaza frecventa de redare }
  time_constant:=256-1000000 div frequency;
  WriteDSP($40);
  WriteDSP(time_constant);
          { Seteaza tipul redarii (8-biti) }
  WriteDSP($14);
  WriteDSP(Lo(size)); WriteDSP(Hi(size))
end
End.
```

Bibliografie:

1. Mark Feldman - *PCGPE* (PC Games Programming Encyclopedia, format electronic)
2. Creative Labs Corporation - *Sound Blaster Resource Kit*

Adrian Monea este student în anul I la Facultatea de Automatică și Calculatoare, Universitatea Tehnică Cluj.

Extinderea CLASELOR

Horia Georgescu

Limbajul Java a fost conceput ca un limbaj orientat pe obiecte. Am prezentat deja două dintre caracteristicile orientate spre obiecte: încapsularea și legarea dinamică. Vom învăța în continuare despre clase extinse, redefinirea metodelor, ascunderea câmpurilor, metode și clase abstracte.

Reamintim cele două concepte învățate, legate de orientarea spre obiecte:

- *încapsularea* (datele, împreună cu operațiile asupra lor, sunt "puse la un loc", și anume într-o clasă);
- *legarea dinamică*: în aceeași clasă pot exista mai multe metode (respectiv constructori) cu același nume, dar cu signaturi diferite; identificarea metodei (constructorului) a cărei (cărui) signatură corespunde apelului se face în timpul executării programului, deci dinamic.

Definirea claselor extinse

Dar facilitatea cea mai importantă legată de OOP (programarea orientată pe obiecte) este cea relativă la *posibilitatea de extindere a claselor*. Foarte pe scurt, deci incomplet și nu în întregime riguros, aceasta constă în:

- o clasă poate fi extinsă, adăugându-se noi câmpuri și noi metode, care permit considerarea unor trăsături suplimentare (dacă ne gândim la câmpuri) și la operații noi (asupra câmpurilor "inițiale", dar și asupra câmpurilor nou adăugate);
- unele metode ale clasei pe care o extindem pot fi redefinite, iar anumite câmpuri ale aceste clase pot fi "ascunse";
- un obiect având ca tip clasa extinsă poate fi folosit oriunde este așteptat un obiect al clasei de la care am pornit.

Exemplul 1

Să presupunem că dorim să urmărim mișcarea unui punct în plan. Vom începe prin a considera clasa:

```
class Punct {
  int x,y; Punct urm;
  Punct(int x,int y) {
    this.x=x;
    this.y=y;
    urm=null;
  }
  void Origine() { x=0; y=0; }
  Punct Miscare(int x, int y) {
    Punct p=new Punct(x,y);
    this.urm=p;
    return p;
  }
}
```

Clasa `Punct` conține:
- câmpurile `x,y,urm`;
- constructorul `Punct` cu signatura `(int,int)`;
- metoda `Origine` cu signatura `()` și metoda `Miscare` cu signatura `(int,int)`.

Vom extinde clasa `Punct`, astfel încât punctul să aibă și o culoare:

```
class Pixel extends Punct {
  String culoare;
  void Origine() { super.Origine();
                   culoare="alb"; }
  Pixel(int x, int y, String culoare) {
    super(x,y);
    this.culoare=culoare;
  }
}
```

Clasa `Pixel` conține:
- câmpurile `x,y,urm` (*moștenite* de la clasa `Punct`) și `culoare` (ultimul a fost adăugat prin extindere);

- constructorul `Pixel` cu signatura `(int,int,String);`
- metoda `Origine` cu signatura `()` și metoda `Miscare` cu signatura `(int,int)`, cu precizarea că metoda `Origine` a fost modificată (redefinită).

Este adoptată următoarea terminologie: clasa `Punct` este *superclasă* a lui `Pixel`, iar `Pixel` este *subclasă* (clasă extinsă) a lui `Punct`.

În Java, clasele formează o structură de arbore în care rădăcina este clasa `Object`, a cărei definiție apare în pachetul `java.lang`. Orice clasă extinde direct (implicit sau explicit) sau indirect clasa `Object`.

Termenii de superclasă și subclasă se referă exclusiv la relația *tată-fiu* în arborele de clase. *Este incorect de a interpreta acești termeni în sensul de incluziune* !

În exemplul de mai sus este ușor de ghicit că apelurile `super.Origine()` și `super(x,y)` se referă respectiv la metoda `Origine` și la constructorul din superclasa `Punct` a lui `Pixel`. Vom reveni însă cu toate detaliile legate de **`super`**.

Fie `Sub` o clasă ce extinde clasa `Super`. Sunt importante următoarele precizări:

- obiectele de tip `Sub` pot fi folosite oriunde este așteptat un obiect de tipul `Super`. De exemplu, dacă un parametru al unei metode este de tipul `Super`, putem invoca acea metodă cu un argument de tipul `Sub`;

- spunem că `Sub` *extinde comportarea* lui `Super`;

- o clasă poate fi subclasă a unei singure clase deoarece, reamintim, clasele formează o structură de arbore. Drept consecință o clasă nu poate extinde două clase, deci nu există *moștenire multiplă* ca în alte limbaje (există însă alte mecanisme pentru a include și această facilitate și anume *interfețele*);

- o metodă din superclasă poate fi *modificată* (*redefinită*, *suprapusă*) în subclasă (evident, ne referim aici la o redefinire a metodei folosind aceeași signatură, pentru că, dacă scriem în subclasă o metodă cu signatură diferită de signaturile metodelor cu același nume din superclasă, atunci va fi vorba de o metodă nouă). La redefinire nu putem modifica tipul valorii întoarse de metodă. Accesul la metoda cu aceeași signatură din superclasă se face, după cum vom vedea, prin precalificare cu **`super`**;

- un câmp redefinit în `Sub` ascunde câmpul cu același nume din `Super`;

- câmpurile neascunse și metodele neredefinite sunt automat *moștenite* de subclasă (în exemplul de mai sus este vorba de câmpurile `x` și `y` și de metoda `Miscare`);

- apelurile **`super`** din metodele și constructorii subclasei trebuie să apară ca primă acțiune.

Din nou despre constructori

Suntem acum în măsură să prezentăm mai complet acțiunile întreprinse la crearea unui obiect.

Prima acțiune constă în a atribui câmpurilor valorile inițiale prin lipsă:

`0`	-	pentru tipuri numerice
`\u0000`	-	pentru tipul **`char`**
`false`	-	pentru tipul **`boolean`**
`null`	-	pentru referințe (la obiecte, inclusiv tablouri și **`String`**).

În continuare este invocat constructorul corespunzător (determinat de signatură); acțiunea sa cuprinde trei faze:

1) Invocarea unui constructor al superclasei și anume:
   - explicit, cu **`super`**`(...)` ;
   - implicit: este vorba de constructorul fără argumente.
2) Executarea, în ordinea în care apar, a inițializărilor directe ale câmpurilor;
3) Executarea corpului constructorului.

Observații:

- un constructor poate invoca un alt constructor și anume prin **`this`**`(...)`; această invocare poate avea loc numai ca primă acțiune a constructorului (deci nu pot fi invocați doi constructori). În schimb acest mecanism nu poate fi folosit pentru a invoca un constructor din corpul unei metode;

- un constructor poate invoca o metodă, caz în care se consideră implementarea metodei corespunzătoare *tipului real* al obiectului (vezi paragraful următor);

- dacă din constructorul `Pixel` din exemplul 1 am elimina invocarea **`super`**`(x,y)`, compilatorul ar semnala o eroare, deoarece, conform mecanismului descris mai sus, constructorul `Pixel` va invoca implicit constructorul fără argumente al superclasei, care însă nu există (el există implicit doar dacă nu s-a specificat un alt constructor).

Exemplul 2

Să considerăm clasele:

```
class a { int va=1, v; a() { v=va; }
}

class b extends a  {
  int vb=2; b() { v=va+vb; }}
```

La crearea unui obiect de tipul `b`, prin **`new`** `b()`, au loc în ordine următoarele acțiuni:

- câmpurile `v`, `va`, `vb` primesc valoarea prin lipsă `0`;
- este invocat constructorul `b`;
- este invocat constructorul `a` (al superclasei);
- are loc inițializarea directă a lui `va`, care primește valoarea `1`;

- este executat corpul constructorului a și ca urmare v primește valoarea 1;
- are loc inițializarea directă a lui vb, care primește valoarea 2;
- este executat corpul constructorului b și ca urmare v primește valoarea 3.

Redefinirea metodelor și ascunderea câmpurilor

O metodă declarată într-o clasă poate fi redefinită (suprapusă) într-o subclasă prin declararea ei în subclasă cu același nume, aceeași signatură și același tip al valorii întoarse.

Pot fi redefinite doar metode nestatice. Rezultă că metodele statice sunt finale.

În metoda redefinită putem schimba modificatorul de acces, cu condiția ca dreptul de acces să crească; reamintim că modificatorii de acces, în ordine de la cel mai restrictiv la cel mai larg, sunt: **private**, **protected**, cel implicit (*package*) și **public**.

Câmpurile nu pot fi suprapuse, ci ascunse prin redeclararea lor într-o subclasă. Câmpul cu același nume din superclasă nu mai poate fi accesat direct prin numele său, ci va trebui folosit **super** (vezi paragraful următor) sau o referință la tipul superclasei. Diferența de terminologie va fi explicată mai jos.

Fie A o clasă și fie B o subclasă a sa. Să considerăm următoarele patru acțiuni echivalente din punctul de vedere al obiectului a ce ia naștere:

1) `A a; a=new B(...);`
2) `A a=new B(...);`
3) `A a; B b; b=new B(...); a=b;`
4) `A a; B b; b=new B(...); a=(A) b;`

Să observăm că este vorba de o conversie de la o subclasă la o superclasă, conversie numită *upcasting* ; ea poate fi implicită (ca în primele trei acțiuni) sau explicită (ca în cea de a patra acțiune).

Vom spune că obiectul a are:
- *tipul real* B;
- *tipul declarat* A.

Tipul real al unui obiect coincide cu tipul său declarat (este cazul obiectului b) sau este o subclasă a tipului declarat (vezi obiectul a).

Fie camp un câmp al clasei A, ce este redeclarat (ascuns) în subclasa B. Dacă obiectul a face referire la câmpul camp, atunci este vorba de câmpul declarat în clasa A, adică este folosit tipul declarat al obiectului.

Fie met o metodă a clasei A, care este redefinită în subclasa B. La invocarea metodei met de către obiectul a, este folosită implementarea corespunzătoare metodei redeclarate, adică este folosit tipul real al obiectului.

Exemplul 3

Considerăm o formă nouă a claselor din exemplul 2:

```
class A {
  int va=1, v;
  A() { v=va; System.out.print(""+met()); }
  int met() { return va; }
}

class B extends A  {
  int vb=2, v;
  B() { v=va+vb; }
  int met() { return vb; }
}

class constr {
  public static void main (String[] s) {
    A a=new B();
    System.out.println("\t"+a.met());
  }
}
```

Rezultatele tipărite de programul de mai sus sunt 0 și 2, deoarece:
- la invocarea metodei met din cadrul constructorului clasei A este folosit tipul real al obiectului, adică este întors rezultatul furnizat de metoda met redeclarată în B; la momentul invocării, valoarea curentă a lui vb este 0 (vezi paragraful precedent);
- invocarea metodei met în metoda principală, realizată de obiectul a, se va referi tot la metoda met din clasa B.

De ce este însă folosit un tip (cel declarat) pentru câmpuri, și alt tip (cel real) pentru metode? Ideea urmărită este ca a să "preia" cât mai multe informații din tipul A. Obiectul a are următorii membri: câmpurile v, va, vb și metoda met. Conform ideii de mai sus, pentru singurul câmp "în litigiu" și anume v, este ales tipul A, deci tipul declarat. Dar pentru metode nu este nimic de ales: va fi folosit tipul real, deoarece ele pot conține referiri și la noile câmpuri introduse în clasa B (în cazul nostru câmpul vb).

Exemplul 4

Să considerăm următorul program:

```
class A {
  String s="Super";
  void scrie() {
    System.out.println("A : "+s);
  }
}

class B extends A {
  String s="Sub";
  void scrie() {
    System.out.println("A : "+s);
  }
}
```

```
class Ex4 {
  public static void main(String[] s) {
    B b = new B(); A a = b;
    a.scrie(); b.scrie();
    System.out.println(a.s + "\t" + b.s);
  }
}
```

La executare sunt tipărite următoarele:

```
B : Sub
B : Sub
Super Sub
```

adică rezultatele așteptate, ținând cont că:
- obiectul `b` are tipul `B` atât ca tip declarat, cât și ca tip real;
- obiectul `a` are tipul declarat `A`, iar tipul real este `B`.

Considerațiile de mai sus referitoare la noțiunile de tip declarat și tip extins conduc și la concluzia că o conversie de la o superclasă la o subclasă nu pot fi totdeauna acceptate. Este suficient să considerăm exemplul de mai sus pentru a realiza că o secvență de tipul:

```
A a; B b; a=new A(); b=a;
```

nu este acceptabilă, deoarece de exemplu clasa `B` poate defini noi câmpuri.

O conversie de la o superclasă la o subclasă (numită *downcasting*) este permisă numai dacă este făcută explicit și numai dacă referința la superclasă este de fapt o referință la subclasă. Iată cadrul tipic în care se realizează acest lucru (în continuare `A` este superclasă a lui `B`):

```
A a; B b1,b2;
b = new B(); . . .
a = b; . . .
b1 = (B) a;                    (1)
```

Menționăm că putem verifica dacă un obiect `Ob` este de un anumit tip `Tip` sau poate fi convertit la acel tip, folosind cuvântul cheie **instanceof** în construcția:

```
Ob instanceof Tip
```

care întoarce o valoare booleană. Referința **null** nu este de nici un tip, deci:

```
null instanceof Tip
```

este `false`, indiferent de tipul `Tip`.

De exemplu instrucțiunea:

```
System.out.print((b1 instanceof A)+
                 "\t"+(b1 instanceof B));
```

plasată după (1), produce la ieșire:

```
true  true
```

Vom discuta despre conversii într-un articol aparte.

Cuvântul cheie super

Putem folosi cuvântul cheie **super** în orice metodă nestatică a unei clase extinse, și anume sub una dintre formele:
- **super**`(...)` : pentru a invoca un constructor al superclasei clasei curente;
- **super**`.met(...)` : pentru a invoca o metodă a superclasei, metodă ce a fost redefinită în clasa curentă;
- **super**`.c` : pentru a accesa un câmp `c` al superclasei, câmp ce a fost ascuns în clasa curentă prin redefinirea sa;
- **super**`.finalize()` : pentru a controla activitatea colectorului de gunoaie; amânăm prezentarea acestei facilități.

Observăm că la accesarea unui câmp sau a unei metode, **super** acționează ca referință la obiectul curent ca instanțiere a superclasei sale.

Pentru exemplificare, să considerăm programul:

```
class A { String s="Super";
  void scrie() {
    System.out.println("A:"+s);
  }
  void metoda() {
    System.out.println("A:");
  }
}

class B extends A {
  String s="Sub";
  void scrie() {
    System.out.println("B:"+s+" "+super.s);
  }
  void metoda() {
    scrie();
    super.scrie(); super.metoda();
  }
}

class AB {
    public static void main(String[] x) {
    B b = new B(); b.metoda();
    System.out.println(b.s);
    System.out.println("****************");
    A a; a = b; a.metoda();
    System.out.println(a.s);
    System.out.println("****************");
    A c = new A(); c.metoda();
    System.out.println(c.s);
  }
}
```

care produce la ieșire următorul rezultat:

```
B : Sub Super
A : Super
A:
Sub
****************
B : Sub Super
A : Super
A:
Super
****************
A:
Super
```

Construcțiile de forma `b.super.metoda()` (care ar intenționa ca pentru obiectul `b` să invoce metoda `metoda` din superclasa clasei de tipul căreia este) sau urcarea mai multor niveluri în arborele de clase prin `super.super` sunt incorecte, neîncadrându-se în formele prezentate anterior de folosire a lui **super**. Pentru a avea acces la un câmp ascuns, aflat cu două niveluri mai sus în arborele de clase, putem proceda de exemplu astfel:

```
class A {
  int x=1;
  void met() {
    System.out.print("\t"+x);
  }
}
class B extends A {
  boolean x=false;
  void met() {
    System.out.print("\t"+x); super.met();
  }
}

class C extends B {
  double x=3.14;
  void met() {
    System.out.print("\t"+x); super.met();
  }
}
class subsup {
  public static void main (String[] s) {
    C Ob=new C(); Ob.met();
  }
}
```

prin a cărui executare obținem la ieșire: `3.14  false  1`

Din nou despre modificatori

Reluăm discuția despre modificatori, pentru a include și aspectele legate de extinderea claselor.

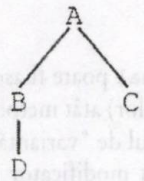
Reamintim că modificatorii se împart în 4 categorii:
- modificatorii de acces (**public**, **protected**, **private** și cel implicit);
- modificatorul **static**;
- modificatorul **synchronized**;
- modificatorul **final**.

Modificatorul **synchronized** va fi studiat separat atunci când vom vorbi de *fire de executare*.

Legat de modificatorul **static** adăugăm la cele deja cunoscute că o metodă statică a unei clase nu poate fi redefinită într-o subclasă, deci este finală. În schimb un câmp static poate fi redefinit (ascuns) într-o subclasă. Evident, nici metodele declarate cu **final** nu pot fi redefinite în subclase.

Referitor la modificatorii de acces, elementele noi ce intervin la extinderea claselor se referă la moștenirea lor în subclase (vezi tabelul). Precizăm că problema moștenirii se pune (bineînțeles) numai pentru acei membri ai clasei care nu sunt statici.

Precizăm efectul lui **protected**: un membru declarat cu **protected** poate fi accesat dintr-o clasă numai prin intermediul unor obiecte al căror tip este un descendent al clasei în arborele de clase. Să considerăm următorul subarbore de clase:



Presupunem că în clasa `A` apare un câmp `x` declarat cu **protected**. Să presupunem că din `B` facem o referire la acest câmp prin `c.x`, unde `c` este un obiect de tipul `C`. Această referire nu este permisă (exceptând cazul în care clasele `B` și `C` apar în același pachet) deoarece ieșim din subarborele din care se face referirea.

Restricția de mai sus nu mai funcționează pentru membrii declarați atât cu **protected**, cât și cu **static**: pentru orice metodă din `B,C,D` putem accesa câmpul `x` prin referințe având oricare dintre tipurile `A,B,C,D`.

Moștenirea în subclase

Modificator	Membrii clasei cu acest modificator sunt accesibili...	Membrii clasei cu acest modificator...
public	de oriunde clasa este accesibilă	sunt moșteniți în subclase
protected	din cod din același pachet și din subclasele clasei respective	sunt moșteniți în subclase
implicit (package)	din cod din același pachet	sunt moșteniți numai în subclase din același pachet
private	numai din interiorul clasei	nu sunt moșteniți în subclase

Și constructorilor le putem atașa modificatori de acces. Un caz limită este cel în care într-o clasă C există cel puțin un constructor și toți constructorii sunt declarați cu **private**. În acest caz nu putem construi direct din afara clasei un obiect având ca tip această clasă. Astfel de clase sunt folosite de obicei pentru a "stoca" metode și câmpuri statice, un exemplu fiind clasa System. Dar aceasta nu înseamnă că nu putem folosi o construcție indirectă: din afara clasei putem invoca de exemplu o metodă statică a lui C ce construiește un obiect de tipul C și îl întoarce ca rezultat:

```
class C {
  private C() {}
  static C met() {
    return new C();
  }
  void scrie() {
    System.out.println("O.K.");
  }
}

class constrf {
  public static void main (String[] s) {
    C Ob = C.met();
    Ob.scrie();
  }
}
```

Modificatorul **final** poate fi asociat (în afara variabilelor locale și câmpurilor) atât metodelor cât și claselor. El trebuie înțeles în sensul de "variantă finală":

- o metodă cu acest modificator nu poate fi redefinită într-o subclasă;
- o clasă cu acest modificator nu poate fi extinsă.

Metodele declarate cu **private** sau/și **static** sunt echivalente cu **final**, iar din punctul de vedere al redefinirii, nu pot fi redefinite.

Dacă o metodă este finală, "ne putem baza" pe implementarea ei (bineînțeles dacă nu invocă metode nefinale). În unele situații, este bine să marcăm toate metodele cu **final**, dar nu și clasa, pentru a permite extensii (subclase) ale ei.

În articolul dedicat accesului și vizibilității, la determinarea semnificației unui identificator, am rămas datori cu explicarea celei de a patra căutări: ea are în vedere luarea în considerație a membrilor clasei, inclusiv a membrilor moșteniți de la superclase.

Metode și clase abstracte

Se întâmplă frecvent ca atunci când lucrăm cu clase extinse, pentru o clasă să nu putem preciza implementarea unei metode, deoarece în subclasele sale ea va fi specifică fiecăreia dintre ele. Atunci metodele din această categorie vor fi marcate cu cuvântul cheie **abstract** și se vor reduce la antetul lor; în același timp și clasa trebuie însoțită de atributul **abstract**. Fiecare metodă abstractă trebuie implementată, adică (re)definită în orice subclasă care nu este la rândul său abstractă.

Nu este posibil să creăm obiecte ca instanțe a unei clase abstracte. Pe de altă parte, într-o subclasă putem redefini o metodă a superclasei transformând-o într-o metodă abstractă; aceasta are sens, de exemplu, dacă în subarborele având ca rădăcină subclasa, comportamentul materializat prin acea clasă nu mai este valabil în subarbore și este specific fiecărei extensii a subclasei.

Exemplu

Următoarea clasă are ca scop să măsoare timpul necesitat de executarea unei metode oarecare fără parametri și care nu întoarce vreun rezultat:

```
abstract class C {
  abstract void met();
  public long timp() {
    long t0 = System.currentTimeMillis();
    met();
    return System.currentTimeMillis()-t0;
  }
}
```

unde a fost folosită metoda statică currentTimeMillis() a clasei System. Într-o clasă ce extinde C, se poate (re)defini metoda met și apoi, folosind un obiect ce este o instanțiere a noii clase, putem apela metoda timp pentru a determina timpul de executare a metodei met, ce are acum o implementare precisă. După cum îi sugerează numele, metoda currentTimeMillis() întoarce timpul în milisecunde:

```
class C1 extends C {
  void met () {
    int x=0;
    for (int i=0; i<1000000; i++) x=x+1-1;
  }
}

class abstr {
  public static void main(String[] s) {
    C1 Ob=new C1();
    System.out.println( Ob.timp() );
  }
}
```

Legat de exemplul de mai sus, nu trebuie să ne mire că la executări diferite vom obține timpi diferiți: aceasta este o consecință a faptului că interpretorul Java lucrează sub sistemul Windows (chiar dacă a fost lansat din DOS) și pe perioada executării programului nostru, sistemul "mai face și altceva".

Prof. dr. Horia Georgescu este cadru didactic la Facultatea de Matematică din București, poate fi contactat prin e-mail la adresa: hg@colegiu.cs.unibuc.ro

Pagini WEB cu HTML 3.2

Mihai Scorțaru

În sfârșit am ajuns într-o fază în care cunoștințele acumulate vă permit să realizați pagini WEB complexe. Sperăm că acestea vă vor satisface în calitate de creatori ale acestora, iar "vizitatorii" se vor bucura și ei.

Conținutul unei pagini Web "profesionale"

În cele ce urmează vor fi prezentate câteva modalități de a controla spațiul dintre texte și imagini. Cunoașterea acestor metode este esențială pentru ca paginile dumneavoastră să fie atractive și ușor de citit.

Fiecare aspect al unei pagini Web trebuie să reflecte scopul pentru care a fost creată. Textul și imaginile trebuie să comunice mesajul care se dorește a fi transmis cititorilor. Modul în care acestea sunt îmbinate are un impact mare asupra ideii pe care și-o va face cititorul despre pagina respectivă. În continuare vă prezentăm *câteva restricții care ar trebui respectate în crearea unor pagini Web*. Este evident că unele pagini, cu un conținut special, nu vor putea respecta, în totalitate, aceste reguli.

Număr de cuvinte: Se recomandă să fie între 100 și 500 de cuvinte pe pagină.

Întreruperea textului: La fiecare 40-100 de cuvinte trebuie să apară un titlu, o linie, sau o imagine; articolele lungi fac excepție.

Mărimea unei pagini: O pagină trebuie să conțină două până la patru ecrane (la o rezoluție de 640x480 pixeli).

Dimensiunea fișierelor: Fișierele care trebuie încărcate pentru vizualizarea unei pagini nu trebuie să depășească 50 KB (inclusiv imaginile); dacă pagina conține GIF-uri animate sunt recomandate dimensiuni de până la 100 KB.

Viteza: Primul ecran și imaginile cheie trebuie să apară în mai puțin de 5 secunde (dacă se folosește un modem la 56 Kbps).

Culori: Trebuie să existe două până la patru culori dominante.

Fonturi: Nu trebuie să existe mai mult de 3 fonturi (inclusiv cele care apar, eventual, în imagini).

Spații albe: Fundalul trebuie să ocupe cel puțin 75% din pagină.

Contrastul: Nici o culoare a background-ului nu trebuie să fie apropiată de o culoare a textului.

Stil: Textul și imaginile trebuie să fie în concordanță cu tema paginii.

Impresia generală: Pagina, ca un întreg, trebuie să pară echilibrată și atractivă.

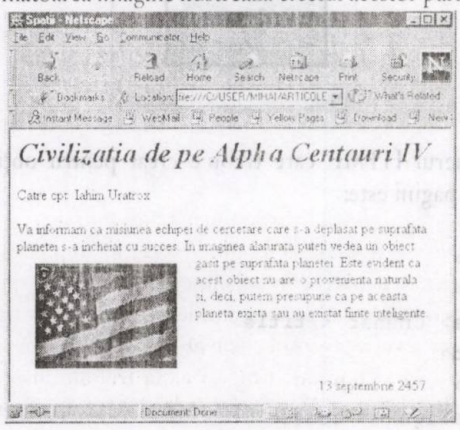
Spațierea și punerea în chenar a imaginilor

Pentru a lăsa loc liber între diferite elemente ale unei pagini se recomandă folosirea unor imagini total transparente.

Dacă o imagine este "înfășurată" de text (s-a folosit `<IMG ALIGN="right">` sau `<IMG ALIGN="left">` atunci, prin folosirea parametrului `CLEAR` al *tag*-ului `<BR>` textul care urmează va fi afișat sub imaginea respectivă. Valorile parametrului `CLEAR` pot fi *right*, *left* sau *all*. Dacă parametrul `CLEAR` are valoarea *right* atunci textul va apărea sub ultima imagine inclusă anterior, aliniată la dreapta, lucrurile fiind similare pentru valoarea *left*. Dacă valoarea este *all* atunci textul va apărea sub ultima imagine inclusă anterior, indiferent de orientarea acesteia.

Poate fi adăugat spațiu suplimentar în jurul imaginilor prin folosirea parametrilor `HSPACE` și `VSPACE`. De exemplu comanda `<IMG SRC="imagine.gif" HSPACE=20 VSPACE=5>` va determina afișarea imaginii cu adăugarea a 20 de pixeli "albi" la stânga și la dreapta, și 5 pixeli "albi" deasupra și sub imagine.

Următoarea imagine ilustrează efectul acestor parametri:



Fișierul HTML corespunzător este:

```
<html>
<head>
<title> Spatii </title>
</head>
<body>
<div align = "center">
<h1> <i>
Civilizatia de pe Alpha Centauri IV
</h1> </i> </div>
<div align = "left">
Catre cpt. Iahim Uratrox, <br> <br>
Va informam ca misiunea echipei de cercetare
care s-a deplasat pe suprafata planetei s-a
incheiat cu succes.
<img src='steag.jpg' align="left" vspace=10
hspace=20 height=120 width=160>
In imaginea alaturata puteti vedea un obiect
gasit pe suprafata planetei. Este evident ca
acest obiect nu are o provenienta naturala si,
deci, putem presupune ca pe aceasta planeta
exista sau au existat fiinte inteligente.
<br clear = "left"> </div>
<div align = "right">
13 septembrie 2457 </div>
</body>
</html>
```

Există posibilitatea ca o imagine să fie afișată într-un chenar. Pentru aceasta trebuie folosit parametrul **BORDER** al *tag*-ului **<IMG>**. Imaginea va apărea într-un chenar de grosime (în pixeli) egală cu valoarea transmisă parametrului **BORDER**. Următoarea imagine prezintă efectul acestui parametru:

Fișierul HTML care trebuie creat pentru obținerea acestei pagini este:

```
<html>
<head>
<title> Chenar </title>
</head>
<body>
<div align = "center">
Valoarea acestui tablou depaseste suma de
200.000 USD.
<img src='forest.jpg' height=120 width=160
border=5 vspace=10> </div>
</body>
</html>
```

Caracteristicile liniilor orizontale

Comanda **<HR>** are câțiva parametri care permit setarea unor caracteristici pentru liniile orizontale. Acești parametri sunt: **SIZE, WIDTH, ALIGN** și **COLOR**.

Valoarea parametrului **SIZE** indică înălțimea liniei (în pixeli).

Parametrul **WIDTH** poate lua ca valori un număr și atunci reprezintă lățimea liniei în pixeli, sau un procent și atunci indică faptul că linia va ocupa acel procent din lățimea ferestrei în care este afișată pagina.

Parametrul **COLOR** permite setarea culorii liniei. Acest parametru are efect numai în Internet Explorer, iar Netscape Navigator îl ignoră.

Alinierea paragrafului curent afectează și alinierea liniei orizontale în cazul în care parametrul **ALIGN** al *tag*-ului **<HR>** lipsește. Astfel, dacă paragraful este aliniat la dreapta, linia se termină la marginea dreaptă a ferestrei, dacă paragraful este aliniat la stânga, linia începe la marginea stângă, iar dacă paragraful este centrat atunci linia va fi centrată.

Image Maps

Uneori este nevoie ca o imagine să poată fi folosită ca legătură spre mai multe pagini diferite. Mai bine zis am vrea ca dacă "apăsăm" pe o anumită regiune a imaginii să fie încărcată o pagină Web, iar dacă "apăsăm" pe o altă regiune să fie încărcată o altă pagină Web. Aceste imagini sunt numite imagini-hartă (*image maps*).

Pentru ca o imagine să fie considerată o hartă, trebuie ca *tag*-ul **<IMG>** să primească parametrul **USEMAP**. Acest parametru va indica harta care va fi folosită pentru imaginea respectivă. Pentru a crea o hartă se folosește un nou *tag* HTML numit **<MAP>**. Acest *tag* are un parametru numit **NAME** care permite denumirea hărții. Între *opening tag*-ul și *closing tag*-ul acestei comenzi trebuie definite regiunile de pe hartă care corespund fiecărei pagini la care se dorește să se ajungă. Pentru definirea unei regiuni trebuie folosit *tag*-ul **<AREA>** care are parametrii **SHAPE, COORDS** și **HREF**.

Parametrul **SHAPE** poate lua valorile *rect*, *poly* și *circle* și indică forma regiunii (dreptunghi, poligon sau cerc).

Valorile parametrul **COORDS** depind de valorile parametrului **SHAPE**. Pentru dreptunghiuri el trebuie să conțină patru numere, reprezentând coordonatele (în pixeli) colțurilor stânga-sus și dreapta-jos ale dreptunghiului. Pentru poligoane parametrul **COORDS** trebuie să conțină un număr de perechi de numere egal cu numărul de vârfuri ale poligonului. Pentru cercuri acest parametru va conține trei numere, primele două reprezentând coordonatele centrului, respectiv raza acestui cerc.

Iată un fișier HTML care ilustrează folosirea acestei modalități de a crea *image maps*:

```
<html>
<head>
<title> Image Map </title>
<map name = "harta">
<area shape="poly" coords="217, 215, 185, 84,
109, 159" href="page1.html">
<area shape="poly" coords="5, 218, 139, 185,
64, 108" href="page2.html">
<area shape="poly" coords="5, 7, 33, 141, 112,
61" href="page3.html">
<area shape="poly" coords="216, 5, 86, 39,
161, 109" href="page4.html">
<area shape="circle" coords="111, 111, 30"
href="page5.html">
</map>
<a href = "nomaps.html">
<img src="imagemap.gif" width=220 height=220
usemap="#harta" ISMAP> </a>
</body>
</html>
```

Au fost definite patru regiuni în formă de triunghi și una în formă de cerc. Parametrul **ISMAP** al *tag*-ului **<IMG>** indică faptul că întreaga imagine poate fi tratată ca un link și poate fi folosită pentru browser-ele mai vechi care nu permit declasarea de imagini hărți. În acest caz un click pe imagine va avea ca efect afișarea paginii corespunzătoare fișierului **NOMAPS.HTML**.

Tabele

Una dintre cele mai puternice unelte folosite în proiectarea paginilor Web este tabelul. Folosirea tabelelor permite aranjarea textului și a imaginilor pe mai multe linii și coloane.

Pentru a crea un tabel, *tag*-ul HTML corespunzător este **<TABLE>**. Pentru ca celulele tabelului să aibă chenare, se folosește parametrul **BORDER** cu ajutorul căruia se precizează grosimea (în pixeli) liniilor care formează chenarele.

Pentru a crea o nouă linie în tabel, trebuie folosit *tag*-ul **<TR>**. O linie conține una sau mai multe "celule". Pentru a crea aceste celule trebuie folosit *tag*-ul **<TD>**. O celulă este o regiune dreptunghiulară a ecranului care poate conține text, imagini și aproape orice altceva.

```
<html>
<head>
<title> Tabel </title>
</head>
<body>
<table border = 2>
<tr>
<td> <div align = "center">
Celula aceasta contine numai text.
Celula urmatoare contine numai o imagine.
</div> </td>
<td><img src="scott.jpg" height=140 width=160>
</td>
<td> <div align = "center">
Aceasta celula contine si text <br> si o
imagine. <br>
<img src = "monster.jpg" height = 96 width =
160> </div> </td>
</tr>
</table>
</body>
</html>
```

Efectul este următorul:

Trebuie precizat faptul că un *tag* folosit într-o anumită celulă va avea efect numai în acea celulă, neafectându-le pe celelalte. Tot așa, *tag*-urile folosite în afara unui tabel nu au efect în interiorul tabelului.

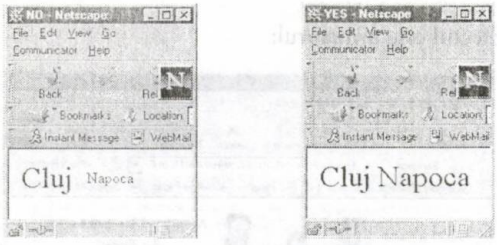
Pentru a ilustra acest lucru, să considerăm fișierele HTML de mai jos:

```
<html>
<head>
<title> NO </title>
<body>
<font size = 5>
<table>
<tr>
<td> <font size = 6> Cluj </td>
<td> Napoca </font> </td>
</tr>
</table>
</font>
</body>
</html>
```

```
<html>
<head>
<title> YES </title>
<body>
<table>
<tr>
<td> <font size = 6> Cluj </font>
</td>
```

```
<td> <font size = 6> Napoca </font>
</td>
</tr>
</table>
</body>
</html>
```

În imaginile de mai jos se observă că în primul caz cuvântul "Napoca" este scris cu litere normale deoarece nici *tag*-ul **<font>** din afara tabelului și nici *tag*-ul **<font>** din prima celulă nu au efect în a doua celulă. Pentru ca "Napoca" să apară cu literele dorite, trebuie folosit *tag*-ul **<font>** așa cum este arătat în al doilea exemplu.

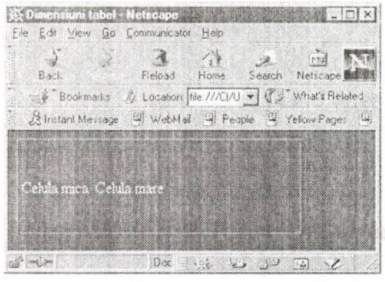


În mod normal o celulă va fi atât de mare cât e nevoie pentru ca elementele ei să încapă în ea. La fel, mărimea tabelului va depinde de mărimea celulelor componente. Există totuși posibilitatea de a defini mărimea unui tabel cu ajutorul parametrilor **HEIGHT** și **WIDTH** ai *tag*-ului **<TABLE>**. Pot fi precizate și dimensiunile fiecărei celule în parte cu ajutorul acelorași parametri, dar transmiși *tag*-ului **<TD>**.

Exemplu:

```
<html>
<head>
<title> Dimensiuni tabel </title>
</head>
<body bgolor = "blue" text = "white">
<table width=200 height=50>
<tr>
<td width=25%> Celula mica </td>
<td width=75%> Celula mare </td>
</tr>
</table>
</body>
</html>
```

Pagina afișată este următoarea:



Conținutul unei celule sau conținutul tuturor celulelor de pe un rând poate fi aliniat atât orizontal cât și vertical folosind parametrii **ALIGN** și **VALIGN** ai *tag*-urilor **<TR>**, respectiv **<TD>**. Valorile parametrului **ALIGN** pot fi *left*, *right* sau *center*, iar cele ale parametrului **VALIGN** pot fi *top*, *bottom* sau *middle*.

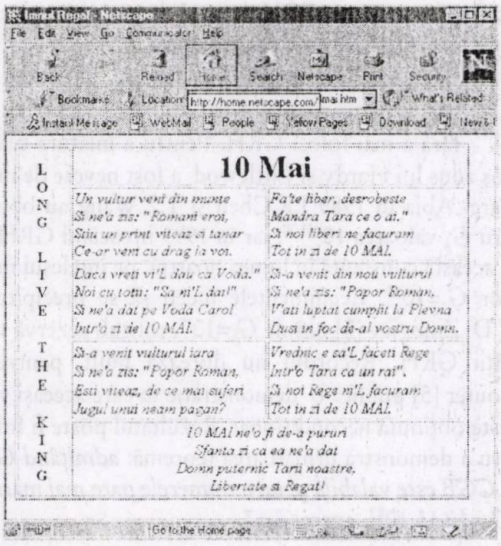
Pentru ca o celulă să ocupe mai multe coloane decât ar fi trebuit în mod normal, poate fi folosit parametrul **COLSPAN**. Pentru ca o celulă să ocupe mai multe linii, *tag*-ul folosit este **ROWSPAN**.

Exemplu:

```
<html>
<head>
<title> Imnul Regal </title>
</head>
<body>
<table border = 2>
<tr>
<td rowspan = 5 align = "center" valign =
"middle" width = 50>
<b> L <br> O <br> N <br> G <p>
L <br> I <br> V <br> E <p>
T <br> H <br> E <p>
K <br> I <br> N <br> G <br> </b>
<td colspan = 2>
<div align = "center">
<font size = 6> <b> 10 Mai </b> </font>
</div>
</td>
</tr>
<tr>
<td> <i> Un vultur veni din munte <br>
Si ne'a zis: "Romani eroi, <br>
Stiu un print viteaz si tanar <br>
Ce-ar veni cu drag la voi. </i> </td>
<td> <i> Fa'te liber, desrobeste <br>
Mandra Tara ce o ai." <br>
Si noi liberi ne facuram <br>
Tot in zi de 10 MAI. </i> </td>
</tr>
<tr>
<td> <i> Daca vreti vi'L dau ca Voda." <br>
Noi cu totii: "Sa ni'L dai!" <br>
Si ne'a dat pe Voda Carol <br>
Intr'o zi de 10 MAI. </i> </td>
<td> <i> Si-a venit din nou vulturul <br>
Si ne'a zis: "Popor Roman, <br>
V'ati luptat cumplit la Plevna <br>
Dusi in foc de-al vostru Domn. </i> </td>
</tr>
<tr>
<td> <i> Si-a venit vulturul iara <br>
Si ne'a zis: "Popor Roman, <br>
Esti viteaz, de ce mai suferi <br>
```

```
Jugul unui neam pagan? </i> </td>
<td> <i> Vrednic e sa'L faceti Rege <br>
Intr'o Tara ca un rai". <br>
Si noi Rege ni'L facuram <br>
Tot in zi de 10 MAI. </i> </td>
</tr> </tr>
<td colspan = 2 align = "center">
<i> 10 MAI ne'o fi de-a pururi <br>
Sfanta zi ca ea ne'a dat <br>
Domn puternic Tarii noastre, <br>
Libertate si Regat! </i> </td>
</tr>
</table>
</body>
</html>
```

Efectul este următorul:



Trebuie precizat faptul că toți membrii redacției respectă Constituția României, ca urmare și forma de guvernământ actuală. Ne cerem scuze față de cei care nu doresc să-și amintească unele lucruri din istoria României. În rest, sperăm că poezia am reprodus-o exact.

Un tabel, o linie sau o celulă poate avea un background diferit de alte background-uri care pot apărea în pagină. Pentru aceasta trebuie folosiți parametrii **BGCOLOR** și **BACKGROUND** ai *tag*-urilor **<table>**, **<TR>**, respectiv **<TD>**. Poate fi controlat și spațiul din jurul marginilor tabelului prin folosirea parametrilor **CELLPADDING** și **CELLSPACING** ai *tag*-ului **<TABLE>**. Cu ajutorul parametrului **CELLSPACING** poate fi setat spațiul dintre marginile tabelului și dintre celule. Parametrul **CELLPADDING** permite setarea spațiului care va înconjura elementele unei celule.

Fișierul HTML următor ilustrează folosirea acestor parametri:

```
<html>
<head>
<title>Fundal</title>
</head>
<body bgcolor = "D0D080">
<table border=1 cellpadding=10 cellspacing=5>
<tr>
<td bgcolor = "#FF8080"> <b> AGORA </b> </td>
<td bgcolor = "#A0FFA0"> <b> AGORA </b> </td>
<td bgcolor = "#C0C0FF"> <b> AGORA </b> </td>
</tr>
<tr>
<td bgcolor = "#80FF80"> <b> AGORA </b> </td>
<td bgcolor = "#A0A0FF"> <b> AGORA </b> </td>
<td bgcolor = "#FFC0C0"> <b> AGORA </b> </td>
</tr>
<tr>
<td bgcolor = "#8080FF"> <b> AGORA </b> </td>
<td bgcolor = "#FFA0A0"> <b> AGORA </b> </td>
<td bgcolor = "#C0FFC0"> <b> AGORA </b> </td>
</tr>
</table>
</body>
</html>
```

Efectul poate fi văzut în imaginea următoare:

După cum am mai spus componentele unei celule pot fi aproape orice. Există posibilitatea ca o celulă să conțină la rândul ei un tabel sau chiar mai multe. Pentru a realiza acest lucru ajunge ca *tag*-ul **<TABLE>** să apară între *opening tag*-ul și *closing tag*-ul comenzii **<TD>**.

Atenție!

Folosirea tabelelor nu este recomandată decât atunci când este absolut necesară. Încărcarea unui tabel durează relativ mult, deoarece la început trebuie calculată mărimea tuturor componentelor tabelului (și trebuie încărcate fișierele corespunzătoare dacă apar imagini) înainte ca o parte a tabelului să poată fi afișată. Acest interval de timp poate fi micșorat dacă se precizează lățimea și înălțimea fiecărei componente folosind parametrii **WIDTH** și **HEIGHT** ai diferitelor *tag*-uri.

Mihai Scorțaru este student la Universitatea Tehnică din Cluj. Poate fi contactat prin e-mail la adresa: scort@licj.soroscj.ro

Conjectura lui GOLDBACH

Mihai Cipu

Teoria numerelor este pentru matematicieni ceea ce este poezia pentru restul muritorilor - la o anumită vârstă, nimeni nu a scăpat de farmecul său. Chiar dacă ulterior i se acordă mai puțină atenție, orice reîntâlnire este prilej de bucurie.

În 1742, *Goldbach* a afirmat că ***orice număr par mai mare sau egal cu 4 este suma a două numere prime.*** Enunț, de o claritate și simplitate într-o atrăgătoare opoziție cu dificultățile de a găsi justificare convingătoare. Eforturi îndelungate, muncă asiduă pentru a stabili rezultate parțiale, lipsite de eleganța conferită însăși de simplitatea enunțului originar. Acest enunț este desemnat în jargonul de specialitate *Conjectura Goldbach Binară* (*CGB*). Pare ciudat, dar primele rezultate în legătură cu această problemă au fost obținute cu rafinate metode analitice. Iar primul în ordine cronologică, sună destul de straniu, având în vedere ceea ce se cere: în 1930 *Șnirelman* demonstrează că orice număr par este suma a cel mult 300000 de numere prime!

O altă abordare constă în a demonstra validitatea conjecturii pentru numere suficient de mari. Cum nici această idee nu a dat roade prea ușor, s-a modificat enunțul inițial. Astfel, în 1923 G.H.Hardy și L.E.Littlewood au dovedit că există un număr G_3 începând de la care întregii impari sunt decompozabili în 3 numere prime. Conjectura Goldbach Ternară CGT, formulată la aproape două secole după Goldbach, susține că G_3=7. Deoarece dacă adunăm 3 la un număr par mai mare sau egal decât 4 se obține un număr impar mai mare sau egal decât 7, este evident că CGT este consecință imediată pentru CGB.

Demonstrației date de Hardy și Littlewood i se pot reproșa două lucruri: *este neefectivă* (nu dă nici un indiciu privind ordinul de mărime pentru G_3) și *condiționată de așa-numita ipoteză generalizată a lui Riemann GRH*. Nu este locul potrivit pentru o formulare a acestei conjecturi, doar amintim că este cel mai dorit rezultat în matematică. Fără a se bucura de popularitatea Marii Teoreme a lui Fermat, este cu mult mai importantă decât aceasta. Domeniul matematicii cunoscut astăzi sub titulatura de *teoria numerelor* constă din demonstrații care invocă GRH și din demonstrații care nu au nevoie de această ipoteză. Rezultatele care se bazează pe GRH sunt mult mai puternice, iar constantele care se obțin sunt mai bune.

În 1937, I.M.Vinogradov stabilește existența lui $G_3=3^{3^{15}}$ fără a mai folosi GRH. Pentru a înlătura celălalt reproș adus lui Hardy și Littlewod a fost nevoie de multă răbdare. Abia în 1989 J.R.Chen și T.Z.Wang au obținut pentru G_3 valoarea 10^{43000}, iar în 1993 invocând GRH, au adus această constantă în lumea noastră, pe înțelesul muritorilor $G_3=10^{50}$. Evenimentele încep să se precipite: în 1997 D. Zinoviev [7] obține $G_3=10^{20}$, tot sub rezerva valabilității GRH. Recordul nu durează mult, pentru că Y.Saouter [5] publică o demonstrație în care aceeași valoare este obținută necondiționat. Rezultatul poate fi folosit pentru a demonstra următoarea teoremă: ***admițând GRH și că CGB este valabilă pentru numerele pare mai mari decât*** $G_2=1{,}615\cdot10^{12}$, ***avem*** $G_3=7$.

Cum rămâne cu problema inițială, CGB? Tot Chen are un rezultat extrem de interesant și surprinzător, anunțat în 1969, dar a cărui demonstrație a putut fi publicată abia după consumarea revoluției culturale: *orice număr par mai mare sau egal cu 4 este suma unui număr prim și a altuia cu cel mult doi divizori primi.* Nici de această dată nu a nimerit ținta în plin. Mai aproape - cel puțin dintr-un punct de vedere, dar și mai departe, dintr-o altă perspectivă - se plasează rezultatul din 1995 al lui O.Ramaré: *orice număr par este suma a cel mult 6 numere prime.* De acum cunoașteți rețeta: invocând GRH, L.Kaniecki are în 1995 o demonstrație pentru următorul enunț: *din GRH și* $G_2 > 1{,}405\cdot10^{12}$, *decurge CGB.* Este clar și următorul pas: estimarea numărului G_2 începând de la care numerele pare sunt obținute adunând două numere prime.

Metodele analitice par a-și fi epuizat resursele; s-a ajuns deja la limita a ceea ce pot aduce ele în problema Goldbach. Se simte nevoia unui ingredient nou, funda-

mental diferit. Acest suflu nou este adus de utilizarea calculatoarelor. Vom prezenta amănunțit modul în care DRS [1] a obținut teoremele de mai jos.

Teorema 1: $G_2 \geq 10^{14}$

Rezultatul este important pentru matematicieni prin consecințele sale:

Teorema 2

Conjectura Goldbach ternară este validă dacă GRH este validă.

Teorema 3

În ipoteza GRH, orice număr par este suma a cel mult 4 numere prime.

Pentru informaticieni mai importante sunt detaliile demonstrației, care este croită pe gustul lor și mai puțin în stilul tradițional al textelor matematice.

Secțiunile următoare sunt structurate după celebra formulă a lui N.Wirth. După ce vom schița ideile folosite pentru a verifica CGB, vom urmări modul în care sunt formulate în program pentru calculator.

Algoritmi pentru a verifica CGB pe un interval [a,b]

Toți algoritmii pe care-i vom prezenta au aceeași structură, bazându-se pe ideea de a găsi mulțimi P și Q constând numai din numere prime astfel încât $P+Q$ acoperă numerele pare din intervalul în care lucrăm. Diferențele încep să se distingă atunci când luăm în considerare modul în care sunt construite asemenea mulțimi.

Sinisalo [6] a folosit $P=\{p_1,p_2,...,p_m\}$ primele m numere prime impare pentru un m convenabil. Pentru fiecare $2e \in [a,b]$ găsește cel mai mic indice t astfel încât $2e-p_t$ este prim. Mulțimea Q constă din numerele prime q cuprinse între $a-d$ și b, unde numărul pozitiv d rămâne a fi determinat printr-un proces de căutare ce folosește arhicunoscutul ciur al lui Eratostene. Se generează un șir crescător de mulțimi: $E_0=\emptyset \subset E_1 \subseteq E_2 \subseteq ...$ după rețeta $E_{i+1}:=E_i \cup (Q+[p_{i+1}])$ pentru $i=0,1,...$ Condiția de terminare este ca E_i să acopere numerele pare dintre a și b. Este ușor de stabilit că parametrul d trebuie să fie mai mare decât cel mai mare număr prim folosit la generarea mulțimilor E_i. La verificarea CGB până la $4 \cdot 10^{11}$ s-a ajuns la $p_{446}=3163$ (acesta este cel mai mic număr prim care, scăzut din 244, 885, 595, 672, dă un număr prim). Dezavantajul acestui algoritm este major: practic trebuie determinate toate numerele prime cuprinse în intervalul de interes.

Pentru stabilirea actualului "record", în [1] este prezentată o altă metodă de abordare. Să examinăm ce modificări apar dacă nu insistăm să obținem cel mai mic număr prim p pentru care $e-p$ este încă prim. Avem nevoie ca P să conțină numere prime impare mai mici decât lungimea $b-a$ a intervalului de interes, iar Q să conste din cele mai mari k numere prime mai mici decât a. Acum parametrul este k, cardinalul mulțimii Q. La fel ca înainte se construiesc mulțimile $F_0:=\emptyset \subset F_1 \subseteq F_2 \subseteq ...$ după rețeta $F_{i+1}:=F_i \cup (P+q_{i+1})$ până se obține că F_t conține toate numerele pare din intervalul $[a, b]$. Mulțimea P depinde de ***lungimea*** intervalului, nu de extremitățile sale. Elementele mulțimii Q depind însă de capătul din stânga a.

Descrierea celor doi algoritmi este relativ imprecisă. În repetate rânduri s-a omis precizarea modalității efective de generare a numerelor prime ce sunt folosite. Pentru o descriere completă a algoritmului se poate invoca ciurul lui Eratostene, care este cunoscut de fiecare. Într-o implementare reală nu va fi însă folosit, fiind mult prea costisitor. Să examinăm mai îndeaproape procedeul alternativ urmat, folosit în [1].

Procedeul ciuruirii se aplică deseori când subiectul căutării nu este direct recunoscut. Începem prin a alcătui o lungă listă de "suspecți" pe care o sortăm eliminând cazurile "evidente" conform informației disponibile în acel moment. Procesul excluderii poate evidenția date noi care influențează decizia la momentul/pasul următor. Când această succesiune de acumulare de indici nu mai produce indicații privind direcția de urmat, procesul se încheie. Obiectele care nu au fost respinse până în acest moment vor fi examinate prin alte mijloace/metode spre a le determina exact identitatea. Aceste idei universal valabile au fost formalizate în aritmetică cu mai mult de 2200 de ani în urmă de Eratostene și sunt nu numai valabile, dar și frecvent folosite în zilele noastre.

Implementarea efectivă

Generarea numerelor prime

Factorizarea este - deocamdată, cel puțin - o problemă foarte mare consumatoare de timp de calcul, ceea ce nu este rău pentru metodele criptografice din familia RSA, dar nici bine pentru celelalte situații în care apar numere prime. Recunoașterea numerelor prime este o cu totul altă problemă, mai ușoară, dar încă nu suficient de rapidă dacă ordinul de mărime al numerelor depășește miliardul. Ciurul lui Eratostene este invocat pentru că este familiar tuturor, dar nu este folosit în programe care prelucrează numere cu peste 10 cifre zecimale. Pentru a stabili dacă un număr este sau nu prim se folosesc metode mult mai rafinate, fundamentate pe rezultate subtile. Schema este următoarea: se caută răspuns la întrebarea "dacă n este prim, ce consecințe decurg?". În cazul în care este posibil de certificat că o astfel de condiție necesară este încălcată, numărul considerat este cu siguranță compus. Dar dacă o îndeplinește, nu avem certitudinea asupra naturii sale. Euristic, este de așteptat ca un număr care trece mai multe teste este prim. Cu cât supraviețuiește mai multor teste, cu atât mai mare este probabilitatea de a fi prim (ca și în sport, de altfel).

În partea elementară a aritmeticii se plasează *Mica Teoremă* a lui Fermat: *pentru orice număr prim p și orice întreg x care nu este multiplu de p, numărul* x^{p-1}-1 *se împarte la p*. Dar această relație poate fi îndeplinită fără ca *p* să fie neapărat prim: spre exemplu, pentru $x=2$ și $p=341=11\cdot31$. Acesta este cel mai simplu dintr-o infinitate de asemenea exemple. Pentru a facilita expunerea, să numim un număr compus *n pseudoprim cu baza x* dacă *n* și *x* nu au factori comuni și *n* divide x^{n-1}-1. De pildă, 561 este pseudoprim cu baza *x* pentru orice *x* nedivizibil prin 3,11 sau 17. Principiul cutiei asigură existența unui exponent *e* mai mic decât *n* astfel încât x^e-1 să fie divizibil prin *n*. Dacă $e=n-1$, atunci matematicienii numesc *n pseudoprim cu baza x*. Dacă *e* se întâmplă să fie cel mai mare divizor impar al lui *n*-1, atunci *n* merită apelativul de *tare pseudoprim cu bază x*. Pentru a vă face idee cât de tare este această proprietate, este suficient să amintim că cel mai mic *n* care se bucură de acest apelativ are 4 cifre, iar până la un milion se întâlnesc doar 46 de impostori care mimează comportamentul numerelor prime și care supraviețuiesc testului reprezentat de *Mica Teoremă* a lui Fermat, fără a fi prime.

În buna tradiție a matematicii făcute *per se*, Jaeschke [9] stabilește că dacă un număr $n<215\ 230\ 289\ 8747$ este tare pseudoprim relativ la baza 2, 3, 5, 7 și 11, atunci *n* este într-adevăr prim. Această proprietate a fost folosită în [1] pentru a testa primalitatea numerelor din mulțimile *P* și *Q*.

Structuri de date

Poezia este atât de dificil de tradus pentru că nu este suficientă utilizarea cuvintelor echivalente după dicționar. Tot așa, transpunerea directă, *mot a mot* a algoritmului matematic într-un program pe înțelesul calculatorului nu are șanse prea mari de reușită datorită irosirii resurselor sistemului de calcul. În cazul concret al algoritmului folosit în [1] dificultatea rezidă în a găsi o reprezentare adecvată în calculator pentru șirul de mulțimi E_i. Caracteristica cea mai importantă a unei asemenea reprezentări este să permită generarea rapidă a termenului următor.

Așa cum am explicat, mulțimea *P* constă din numerele prime impare mai mici decât diferența dintre extremitățile intervalului pentru care se dorește verificarea conjecturii Goldbach. Funcția caracteristică pentru această mulțime este codificată într-un vector de biți ODD de lungime corespunzătoare, cel de-al *i*-lea bit fiind inițializat cu valoarea 1 dacă și numai dacă $2i+1$ este număr prim. Similar, mulțimii E_i i se asociază un vector de biți numit SIEVE, de aceeași lungime cu ODD, în care bitul *i* reprezintă numărul par q_1+2i+1. Inițial ODD este copiat în SIEVE. Ca urmare avem SIEVE[*i*]=1 dacă $2i+1$ este număr prim sau echivalent q_1+2i+1 este suma a două numere prime. Altfel spus, în acest moment inițial SIEVE reprezintă mulțimea E_1. Următoarea acțiune constă în a înlocui vectorul SIEVE cu ceea ce se obține efectuând operația logică *disjuncție* cu vectorul ODD deplasat spre dreapta cu $(q_2-q_1)/2$ poziții. Este simplu de văzut că după efectuarea acestor operații, SIEVE corespunde mulțimii E_2 definite la descrierea algoritmului. În general, E_{i+1} este generat efectuând operația *disjuncție logică* pe fiecare componentă a vectorului SIEVE ce reprezintă E_i, cu vectorul ODD deplasat spre dreapta cu $(q_{i+1}-q_i)/2$ poziții. Conjectura CGB este confirmată dacă la un moment dat toate componentele vectorului SIEVE au valoarea 1.

Configurația utilizată

Verificarea conjecturii Goldbach până la 10^{14} a fost făcută independent de 2 echipe. Una a examinat intervalele $I\times10^{13}$ pentru $I=[2,4]$, $[6,8]$, $[9,10]$ pe supercalculatorul vectorial Cray C916. Pentru restul numerelor s-a folosit un server Power Challenge Array cu 18 unități R8000.

Datorită lungimii vectorului SIEVE, nu s-a verificat îndeplinirea condiției de terminare a execuției, decât după ce șansele de succes au crescut foarte mult. Un raționament strict matematic a condus la concluzia că abia după 170 de pași devine eficientă o astfel de verificare. S-a constatat că doar 4 numere(!) nu erau descompuse în maniera prezisă de Goldbach. După câteva iterații s-a găsit și pentru acești "refractari" descompunerea dorită.

Dar cât a durat execuția acestui program? Pentru a identifica cele aproximativ 430000 numere prime cu cel mult 14 cifre s-au consumat circa 75 ore CPU pentru C916. Generarea mulțimilor E_i a necesitat alte 225 ore. Pentru aceste intervale, mulțimea *Q* a constat din numerele prime cuprinse între $q_1=33835999990007$ și $q_{415}=33836000002499$, acesta din urmă fiind folosit pentru a obține descompunerea lui $e=33836446494106$. Execuția pe rețeaua SGI a însumat 4083 ore CPU, deci în timp real 227 ore.

Bibliografia

1. J.M. Deshouillers, H.J.J.te Riele, Y. Souter, *New experimental results concerning the Goldbach conjecture*, Centrum voor Wiskunde en Informatica Amsterdam, Report MAS-R9804, martie 1994
2. G. Jaeschke, *On strong pseudoprimes to several bases*, Math. of. Comput. 61(1993), 915-926
3. L. Kaniecki, *On Snirelman's Constant under the Riemann hypothensis*, Acta Arithm. 72(1995), 361-374
4. O. Ramare, *On Snirelman's Constant*, Ann.Scuola Norm.Sup.Pissa 22(1995), 647-706
5. Y. Saouter, *Checking the odd Goldbach conjecture up to* 10^{20}, Math of Comput. 67(1998), 863-866
6. M. K. Sinisalo, *Checking the Goldbach conjecture up to* $4\cdot10^{11}$, Math of Comput. 61(1993), 931-934
7. D. Zinoviev, *On Vinogradov's constant in Goldbach's ternary problem*, J. Number Th 65(1997), 334-358

Mihai Cipu este doctor în matematică, cercetător la Institutul de Matematică din cadrul Academiei Române. Poate fi contactat pe email la: mcipu@stoilow.imar.ro

Şi "piticii" CONCUREAZĂ

Groza Sorin

Calculatorul este unealta sau "jucăria" care fascinează copiii încă de la cele mai fragede vârste. Dacă întrebi un copil ce cadou şi-ar dori, cu siguranţă va răspunde "un calculator". Copiii se obişnuiesc repede cu calculatorul şi de multe ori, cu - sau fără îndrumare învaţă să-l utilizeze şi să scrie programe.

Pentru a veni în sprijinul celor care abia acum încep să pătrundă în tainele informaticii, vom prezenta câteva probleme împreună cu algoritmul de rezolvare şi programul sursă.

Am ales pentru început câteva din problemele propuse la ediţiile din 1996 şi 1997 ale concursului de informatică "*INFO-STAR*" organizat de *Clubul Copiilor din Aiud*. Problemele au fost compuse de domnul profesor *Ovidiu Domşa* de la Colegiul "Horea, Cloşca şi Crişan" din Alba Iulia, special pentru acest concurs. Organizatorii îi aduc mulţumiri şi pe această cale. Pentru fiecare problemă prezentăm un algoritm de rezolvare la nivelul copiilor cărora li se adresează.

Trepte

Un pitic vrea să urce o scară care are n trepte de înălţimi date, ordonate crescător. Înălţimile treptelor sunt date în *cm* şi sunt valori întregi. Acolo unde diferenţa între două trepte consecutive este de 1cm piticul urcă fără dificultăţi. Unde diferenţa este mai mare decât 1cm piticul trebuie să ia o pastilă care îi dă putere să sară pe treapta următoare. Cunoscând înălţimile treptelor, prima fiind obligatoriu 0, piticul vrea să afle care este numărul minim de pastile de care are nevoie pentru a urca scara şi de asemenea care este cea mai mare diferenţă dintre două trepte consecutive.

Date de intrare:

```
n=9
0,3,4,6,7,10,19,20,21
```

Date de ieşire:

```
numarul minim de pastile este 4
diferenta maxima este 9
```

Rezolvare

Pentru început se va calcula diferenţa de înălţime pentru fiecare două trepte consecutive. Când aceasta este mai mare decât 1 se va incrementa un contor care în final va reprezenta numărul de pastile necesare piticului. De asemenea, într-o variabilă se va păstra diferenţa; dacă diferenţa curentă este mai mare decât cea mai mare diferenţă găsită până atunci, evident, aceasta se actualizează. Practic, algoritmul determină maximul dintr-o mulţime de numere.

Listing TREPTE.BAS

```
CLS
INPUT "cate trepte"; n
DIM a(n)
FOR i=1 TO n
  PRINT "Inaltimea treptei"; i
  INPUT a(i)
NEXT i
pastile=0
difmax=0
FOR i=2 TO n
  IF a(i)-a(i-1)>1 THEN pastile=pastile + 1
  IF a(i)-a(i-1)>difmax THEN difmax=a(i)-a(i-1)
NEXT i
PRINT "numarul minim de pastile este";pastile
PRINT "diferenta maxima este"; difmax
```

Listing TREPTE.PAS

```
program trepte;
var max,pastile,i,n,a,b:Integer;
Begin
  Write('Introduceti numarul de trepte:');
  Readln(n); pastile:=0; a:=0; max:=0;
  Write('Introduceti inaltimile treptelor: 0');
  for i:=2 to n do
  begin Read(b);
        if b-a>1 then Inc(pastile);
        if b-a>max then max:=b-a;
        a:=b
  end;
  Writeln('Nr. min. pastile', pastile,'.');
  Writeln('Diferenta max. este ', max,'.');
End.
```

Robot

Membrii cercului de electronică de la Clubul Copiilor din Aiud au realizat un robot care știe să se deplaseze la comandă. Astfel, comanda `Nx` deplasează robotul `x` metri spre nord, comanda `Ey`, `Vz`, `St` deplasează robotul `y` metri spre est, `z` metri spre vest, respectiv `t` metri spre sud (`x`, `y`, `z`, `t` sunt numere naturale). Un grup de copii "năzdrăvani" dau o serie de comenzi robotului pentru a-l rătăci. La comanda `STAI` robotul se oprește. Ajutați membrii clubului de electronică ca prin maxim două comenzi să readucă robotul în punctul de plecare.

Date de intrare:

```
N3, V5, S3, E4, E5, N2, STAI
```

Date de ieșire:

```
Robotul revine prin comenzile:
V4
S2
```

Rezolvare

Pentru rezolvarea problemei vom folosi două variabile: `x` și `y` care reprezintă poziția robotului la un moment dat. Inițial ele au valorile egale cu 0, acest lucru reprezentând faptul că robotul se află în originea unui sistem de coordonate. După citirea fiecărei comenzi, `x` și `y` își măresc sau micșorează valoarea cu numărul de pași, în funcție de direcția dată. În final (după citirea comenzii `"STAI"`) `x` și `y` vor reprezenta poziția robotului. În funcție de cadranul în care se află, se vor tipări direcția de revenire și numărul de pași.

Listing ROBOT.BAS

```
CLS
x=0
y=0

1 INPUT "directia"; dir$
IF x$="stai" or x$="STAI" THEN GOTO 10
INPUT "cati pasi"; pasi
IF dir$="N" or dir$="n" THEN x=x+pasi
IF dir$="S" or dir$="s" THEN x=x-pasi
IF dir$="E" or dir$="e" THEN y=y+pasi
IF dir$="V" or dir$="v" THEN y=y-pasi
GOTO 1

10 IF x=0 and y=0 THEN PRINT "OK": END
PRINT "Robotul revine prin comenzile:"
IF y>0 THEN PRINT "V "; y
IF y<0 THEN PRINT "E "; -y
IF x>0 THEN PRINT "S "; x
IF x<0 THEN PRINT "N "; -x
```

Listing ROBOT.PAS

```
program robot;
var dir: string;
    pasi,x,y:Integer;
Begin
  while true do
  begin
    Write('Directia: '); Readln(dir);
    if dir='STAI' then break;
    Write('Numar de pasi: '); Readln(pasi);
    if Upcase(dir[1])='N' then Inc(x,pasi);
    if Upcase(dir[1])='S' then Dec(x,pasi);
    if Upcase(dir[1])='E' then Inc(y,pasi);
    if Upcase(dir[1])='V' then Dec(y,pasi)
  end;
  if (x=0) and (y=0) then Writeln('OK');
  if y>0 then Writeln('V ', y);
  if y<0 then Writeln('E ', -y);
  if x>0 then Writeln('S ', x);
  if x<0 then Writeln('N ', -x)
End.
```

Cuvinte

Se consideră o listă de cuvinte, despărțite printr-un spațiu, introduse de la tastatură. Cuvintele au în componență doar litere mici ale alfabetului latin.

Două cuvinte se *oglindesc* dacă literele unui cuvânt apar în celălalt cuvânt în ordine inversă (*car - rac* se oglindesc).

Un cuvânt este *simetric* dacă citit de la stânga la dreapta și de la dreapta la stânga obținem același cuvânt (*cojoc* este simetric).

a) afișați perechile de cuvinte care se oglindesc;

b) afișați cuvintele simetrice din frază;

c) dacă fraza conține numai cuvinte simetrice sau care se oglindesc să se formeze cea mai lungă frază simetrică (ca număr de cuvinte folosite), fiecare cuvânt fiind folosit o singură dată.

Date de intrare:

```
n=8
la mos era som al cojoc are caiac
```

Date de ieșire:

```
a)
are - era
mos - som
al - la

b)
cojoc
caiac

c)
la mos era cojoc are som al  (o solutie !)
```

Fiecare cuvânt din text se inversează, apoi se caută un cuvânt identic cu inversul; când a fost găsit se tipăresc cele două cuvinte și se depun fiecare în câte un șir. Aceste două șiruri, unul pentru cuvânt și celălalt pentru invers, le vom folosi la punctul c) pentru formarea frazei.

Pentru determinarea cuvintelor simetrice inversăm fiecare cuvânt și verificăm dacă este identic cu cuvântul însuși. În caz afirmativ acesta va fi afișat și depus într-un șir al cuvintelor simetrice, altfel va fi abandonat.

Listing CUVINTE.BAS

```
CLS
INPUT "Cate cuvinte "; n
DIM a$(n): DIM c$(n): DIM r$(n): DIM x$(n)
FOR i=1 TO n
  PRINT "Al"; i; "-lea cuvant este ";
  INPUT a$(i)
NEXT i
k=0
PRINT "a)"
FOR i=1 TO n-1
  b$=""
  FOR t=LEN(a$(i)) TO 1 STEP -1
    b$=b$+MID$(a$(i),t,1)
  NEXT t
  FOR j=i+1 TO n
    IF b$=a$(j) THEN
      PRINT a$(i); "-"; a$(j): k=k+1
      c$(k)=a$(i): x$(k)=a$(j)
    END IF
  NEXT j
NEXT i
IF k=0 THEN
  PRINT "NU SUNT CUVINTE CARE SE OGLINDESC !"
PRINT "b)"
s=0
FOR i=1 TO n
  b$=""
  FOR j=LEN(a$(i)) TO 1 STEP -1
    b$=b$+MID$(a$(i),j,1)
  NEXT j
  IF b$=a$(i)
    THEN PRINT b$: s=s+1: r$(s)=a$(i)
NEXT i
IF s=0
  THEN PRINT "NU SUNT CUVINTE SIMETRICE !"
IF s+k*2=n
THEN
  PRINT "c)";
  fr$=""
  FOR i=1 TO k
    fr$=fr$+c$(i)+" "
  NEXT i
  fr$=fr$+r$(1)+" "
  FOR i=k TO 1 STEP -1
    fr$=fr$+x$(i)+" "
  NEXT i
  PRINT fr$
ELSE
  END
END IF
```

Pentru rezolvarea punctului c) verificăm dacă textul conține doar cuvinte simetrice și care se oglindesc. Acest lucru este adevărat doar dacă numărul cuvintelor depuse în cele trei șiruri este egal cu numărul total de cuvinte (s+2*k=n). În caz afirmativ vom adăuga cuvintele primului șir, urmat de un cuvânt din șirul celor simetrice, apoi cuvintele oglindă ale primului șir, dar de la sfârșit spre început pentru ca fraza să respecte condiția de a fi simetrică.

Listing CUVINTE.PAS

```
program Cuvinte;
var a,c,r,x:array[1..100] of string[25];
    s,i,n,j,k:Integer;
    fraza,b:string[25];
Begin
  Write('Introduceti numarul de cuvinte:');
  Readln(n);
  for i:=1 to n do
  begin Write('Cuvantul ', i, ': ');
        Readln(a[i])
  end;
  Writeln('a)');
  for i:=1 to n-1 do
  begin
    b:='';
    for j:= Length(a[i]) downto 1 do
      b:=b+a[i][j];
    for j:=i+1 to n do
      if b=a[j] then
        begin
          Writeln(a[i], ' - ', a[j]);
          Inc(k); c[k]:=a[i]; x[k]:=a[j]
        end
  end;
  if k=0
  then Writeln('Nu exista perechi de '
              +'cuvinte care se oglindesc!');
  Writeln('b)'); s:=0;
  for i:=1 to n do
  begin
    b:='';
    for j:=Length(a[i]) downto 1 do
      b:= b+a[i][j];
    if b=a[i] then
      begin
        Writeln(b); Inc(s); r[s]:=a[i]
      end
  end;
  if s=0 then Writeln('Nu exista cuvinte '
                      +'simetrice !');
  if s+k*2=n then
    begin
      Writeln('c)');
      for i:=1 to k do
        fraza:=fraza+c[i]+' ';
      fraza:=fraza+r[1]+' ';
      for i:=k downto 1 do
        fraza:=fraza+x[i]+' ';
      Writeln(fraza)
    end
End.
```

Șeful

Un grup format din n copii, așezați în linie la distanțe egale, participă la un joc. Fiecare trebuie să-și aleagă pe rând o sumă de bani distinctă de cele alese de colegii săi. La un moment dat unul dintre copii devine *șef*, iar toți cei din dreapta lui donează suma de bani celui aflat în *stânga șefului* la aceeași distanță față de șef la care se află și el.

Dacă un copil nu are corespondent, rămâne cu suma inițială.

a) verificați dacă fiecare copil alege corect suma;

b) să se determine poziția copilului care va deveni șef și astfel va fi cel mai bogat posibil după donarea banilor de către cei din dreapta lui. Se va afișa poziția șefului, șirul obținut după donație, respectiv poziția și suma celui mai bogat copil.

Date de intrare:

```
n=7
45, 34, 23, 12, 57, 43, 24
```

Date de ieșire:

```
Sume corecte
Sef 3
dupa donatie 102, 46, 23, 0, 0, 43, 24
cel mai bogat 1 cu 102
```

Rezolvare

Problema este foarte interesantă și face apel la procedeul numit "împăturarea unui șir". Primul punct este relativ simplu, el constând în a verifica dacă un element nou introdus nu se regăsește printre elementele introduse până la el. Se vor păstra, apoi afișa pozițiile copiilor care nu verifică această proprietate.

Pentru aflarea șefului se procedează astfel: va fi considerat șef fiecare copil pe rând. Se va realiza împăturarea șirului, adică elementele din dreapta șefului se vor aduna la cele din stânga șefului (respectând simetria față de șef) după care devin nule. Se va determina cel mai bogat copil și poziția lui. Dacă se găsește un copil mai bogat, după fiecare încercare se va reține configurația. În final vom afișa rezultatele pentru situația optimă. Se observă că metoda nu este optimă, ea implementând algoritmul conform căruia se încearcă toate soluțiile și se păstrează cea mai bună. Să nu uităm că problema se adresează "celor mici". Vă propunem să căutați soluții mai performante.

Listing SEFUL.PAS

```
program seful;
uses Crt;

type tip=record
           p:Byte;
           val:Word
         end;
var a,b,c:array[1..100] of Word;
    p:array[1..100] of tip;
    gresit:set of Byte;
    n,i,j,k,sef:Byte;
    t:tip;
    m,max,pmax,psef,pm:Word;
label 11;

Begin
  Clrscr;
  Write('introduceti n '); Readln(n);
  for i:=1 to n do
  begin
    Write('Suma celui de-al',i,'-lea copil ');
    Readln(a[i]);
    for j:=1 to i-1 do
      if a[j]=a[i] then gresit:=gresit+[i]
  end;
  if gresit<>[] then
  begin
    Write('Copiii ');
    for i:=1 to n do
      if (i in gresit) then Write(i,' ');
    Writeln('nu au ales corect suma');
    Write('Nu exista solutie ptr. ');
    Writeln('punctul b'); Halt(0)
  end
  else Writeln('Sume corecte');
  for i:=1 to n do
  begin
    sef:=i; b:=a;
    for j:=sef+1 to n do
    begin
      if sef-(j-sef)>0 then
      begin
        b[sef-(j-sef)]:=b[sef-(j-sef)]+b[j];
        b[j]:=0
      end
    end;
    m:=0;
    for j:=1 to n do
    if b[j]>m
    then begin pm:=j; m:=b[j] end;
    if m>max then
    begin
      c:=b; max:=m; psef:=sef; pmax:=pm
    end
  end;
  Writeln('Sef ',psef);
  Write('Dupa donatie ');
  for i:=1 to n do write(c[i], ' ');
  Writeln('Cel mai bogat ',pmax,' cu ',max);
  Readln
End.
```

Domnul Sorin Groza este profesor la Clubul Copiilor din Arad.

Vă place GInfo?

Nume: _____ Prenume: _____ Vârsta: _____

Adresa: _____

Localitate: _____ Cod poștal: _____ Județ: _____ Telefon: _____

Ocupația: elev(ă) informatică ☐ elev(ă) ☐ student(ă) ☐ cadru didactic ☐ altă ocupație ☐ (precizați: _____)

Școala (firma, organizația etc.): _____

Dotări

Folosiți un calculator acasă? Da ☐ Nu ☐

Câte calculatoare sunt instalate în școală: (aprox.) _____

Aveți acces la e-mail: Da ☐ Nu ☐

Aveți acces la WWW: Da ☐ Nu ☐

Cum apreciați dotarea cu tehnică de calcul a școlii: _____

Aprecieri generale

Acordați note de la 1 la 10 pe următoarele criterii:

Structura tematică: ___

Calitatea articolelor: ___

Prezentare grafică: ___

Calitate tipografică: ___

Considerați că prețul este:

prea mare ☐ accesibil ☐ prea mic ☐

De cât timp cunoașteți Gazeta de Informatică (ani)? _____

Cum ați procurat acest număr:

chioșc ☐ abonament ☐ cadou ☐ altfel ☐

În acest număr

Apreciați cu note de la 1 la 10 articolele acestui număr:

Alegeri prezidențiale ___

Problema rutajului ___

Derandomizare ___

Microprocesoare ___

Programarea plăcii de sunet ___

Extinderea claselor în Java ___

Pagini Web cu HTML 3.2 ___

Conjectura lui Goldbach ___

Și piticii concurează ___

Articolul care v-a plăcut cel mai mult: _____

De ce? _____

Articolul care v-a plăcut cel mai puțin: _____

De ce? _____

Considerați că ponderea problemelor rezolvate este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că ponderea problemelor propuse este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că ponderea subiectelor teoretice este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că nivelul de accesibilitate al articolelor este

prea coborât ☐ potrivit ☐ prea ridicat ☐

În viitor

În următoarele numere așteptați:

mai multe articole de programare? Da ☐ Nu ☐

mai multe articole teoretice? Da ☐ Nu ☐

mai multe articole despre Internet? Da ☐ Nu ☐

mai multe prezentări de programe? Da ☐ Nu ☐

prezentări de jocuri? Da ☐ Nu ☐

articole despre hardware? Da ☐ Nu ☐

V-ar plăcea să citiți:

pagini redactate de elevii unor licee? Da ☐ Nu ☐

lucrări premiate, realizate de elevi? Da ☐ Nu ☐

actualități școlare? Da ☐ Nu ☐

prezentarea unor situri Web? Da ☐ Nu ☐

tips & tricks? Da ☐ Nu ☐

Comentarii:

_____

_____

_____

_____

_____

Dacă nu vă este suficient spațiul, folosiți coli albe suplimentare

Așteptăm scrisorile voastre

Poșta redacției

În acest număr poșta redacției revine și sperăm că de-acum va fi o prezență permanentă în paginile revistei. Dar pentru a fi cât mai interesantă și utilă, e nevoie și de concursul vostru. Deci, scrieți-ne!

Înainte de a trece la scrisorile selectate pentru acest număr, se cuvine să fac câteva precizări legate de "identitatea" semnatarilor mesajelor pe care le primim. Practica jurnalistică a impus o regulă respectată de toate publicațiile serioase: ***nu se răspunde la scrisorile anonime***. Această regulă este valabilă și pentru GInfo. În plus, trebuie să mărturisesc că am suspiciuni privitoare la semnăturile "de grup", motiv pentru care vă sugerez ca numele persoanei care a redactat textul să nu lipsească din mesaj. Faptul că nu întotdeauna voi publica numele semnatarului nu înseamnă că scrisoarea nu a fost semnată.

Atestatul

Acestea fiind spuse, putem aborda prima scrisoare... "de grup" (adresată revistei, dar purtând mențiunea "pentru d-ra Anca Voicu"):

Suntem elevi ai clasei cu profil de informatică de la Liceul Teoretic nr. 1 din Reșița. Știm care este metodologia desfășurării atestatului profesional la profilul informatică, stabilită în luna octombrie și avizată de dl. dir. gen. Gh. Rădulescu, dar am aflat din mass-media că nu este obligatoriu examenul de profil la bacalaureat.

În legătură cu examenul de bacalaureat, vă rugăm să ne răspundeți la următoarele probleme (întrebări):

1) Este posibil ca un elev de la profilul informatică să nu susțină examen la disciplina informatică la bacalaureat și totuși să i se permită să dea celelalte două examene (proba practică și susținerea lucrării din timpul anului), iar în final să obțină atestatul ?

2) Ar trebui făcute precizări în legătură cu limbajul de programare în care se poate realiza lucrarea de specialitate (este permis limbajul FoxPro?)

3) Va fi modificată metodologia actuală pentru anul în curs? Dacă da, până când să așteptăm această modificare.

Vă mulțumim pentru înțelegere.

Există anumite indicii (de pildă, folosirea grafiei "vechi") care mă fac să bănuiesc că mesajul a fost redactat de un profesor. Ceea ce nu e rău, dar am vrea ca acest lucru să fie explicit. Mulțumim d-rei Voicu, inspector de specialitate în cadrul MEN, pentru amabilitatea de a ne comunica următoarele răspunsuri:

1) Nu. Atestatul profesional îl pot primi numai acei elevi care aleg în cadrul examenului de bacalaureat proba de informatică. În ceea ce privește tipul de subiecte pe care le va cuprinde teza de bacalaureat, am transmis precizări pe lista de dialog «Iprofi».

2) Lucrarea de specialitate se poate realiza în orice limbaj. Este o lucrare care îmbină creativitatea, fantezia și cunoștințele în domeniu.

3) Nu se modifică metodologia de desfășurare a atestatului.

Vă doresc succes!

FoxPro

Următoarea scrisoare (semnată în original) vine de la un grup de elevi de la un liceu de specialitate și ridică o problematică asemănătoare:

[...] O a doua problemă care a apărut de curând (și care ne-a impulsionat să scriem acum) este problema atestatului la Info. Este o ciudățenie, în opinia noastră, să faci și proiect, să tragi și biletel și să fii obligat să dai și Bac-ul tot la Info.

De asemenea, obligativitatea examinării la "obiectul" FoxPro este stranie. Se știe acum că această "tehnologie" a fost una de tranziție. De ce suntem obligați să învățăm ceva care probabil nu va mai exista peste 5 ani decât în câteva companii românești obscure? Și de ce nu se învață măcar Visual FoxPro?

Nu s-ar putea să se predea ceva cu caracter mai general, de exemplu SQL sau principiile bazelor de date relaționale, cu aplicări specifice mai "slabe"? În plus, studiul unor asemenea principii ar da o temă de proiect: alcătuirea unui sistem mic de baze de date relaționale.

În fine, problema bazelor de date și mai ales a tehnologiilor relative la acest subiect este obscură la noi. Este mai ales neclară limita dintre tehnologie și aplicație specifică. Am aprecia un articol în GInfo care să trateze această problemă, lămurindu-ne și pe noi.

O mustrare discretă adresată ministerului, dacă ajungeți la concluzia că e necesar, ar putea să ne ajute pe toți.

Cu respect pentru presă,
Clasa 12b, liceul xxx.

În privința atestatului, cred că vom avea ocazia să aflăm mai multe păreri pro și contra. Vom încerca să abordăm tema din mai multe unghiuri. Lista *Dialog* vă stă la dispoziție.

Cât privește subiectul (sau "obiectul"?) FoxPro... discuția e mai lungă. Nici eu nu mă număr printre partizanii bazelor de date de tip Xbase (cum este și FoxPro) dar trebuie să admit că pe calculatoarele cu care sunt dotate școlile din România cu greu s-ar putea rula un SGBD mai pretențios. Nu cred, pe de altă parte, că Visual FoxPro ar fi mai potrivit. În primul rând, pe PC-uri cu mai puțin de 32 MB RAM este imposibil de rulat. În al doilea rând, Visual FoxPro "ascunde" de fapt motorul de baze de date în spatele unui mediu orientat către interfața cu utilizatorul, ceea ce nu corespunde (din punct de vedere didactic) materiei numite "baze de date".

Utilizarea unui SGBD relațional propriu-zis (chiar și a unuia non-comercial, cum este Postgres) ar pune probleme de administrare foarte dificile. Implementarea unei arhitecturi client/server minimală este un efort pe care nu și-l permit nici măcar firmele comerciale, cu atât mai puțin școlile.

În fine, Fox-ul dispune de un subset al limbajului SQL, care poate fi utilizat cu succes pentru a exemplifica principiile bazelor de lucru relaționale. Dacă se evită "navigarea" prin pointeri de fișier și se impun câteva restricții (de pildă unicitatea liniilor), FoxPro poate fi folosit într-o manieră "aproape relațională".

Provocarea pe care mi-o aruncați (scrierea unui articol explicativ despre baze de date) mă onorează... Sper să găsesc timpul necesar.

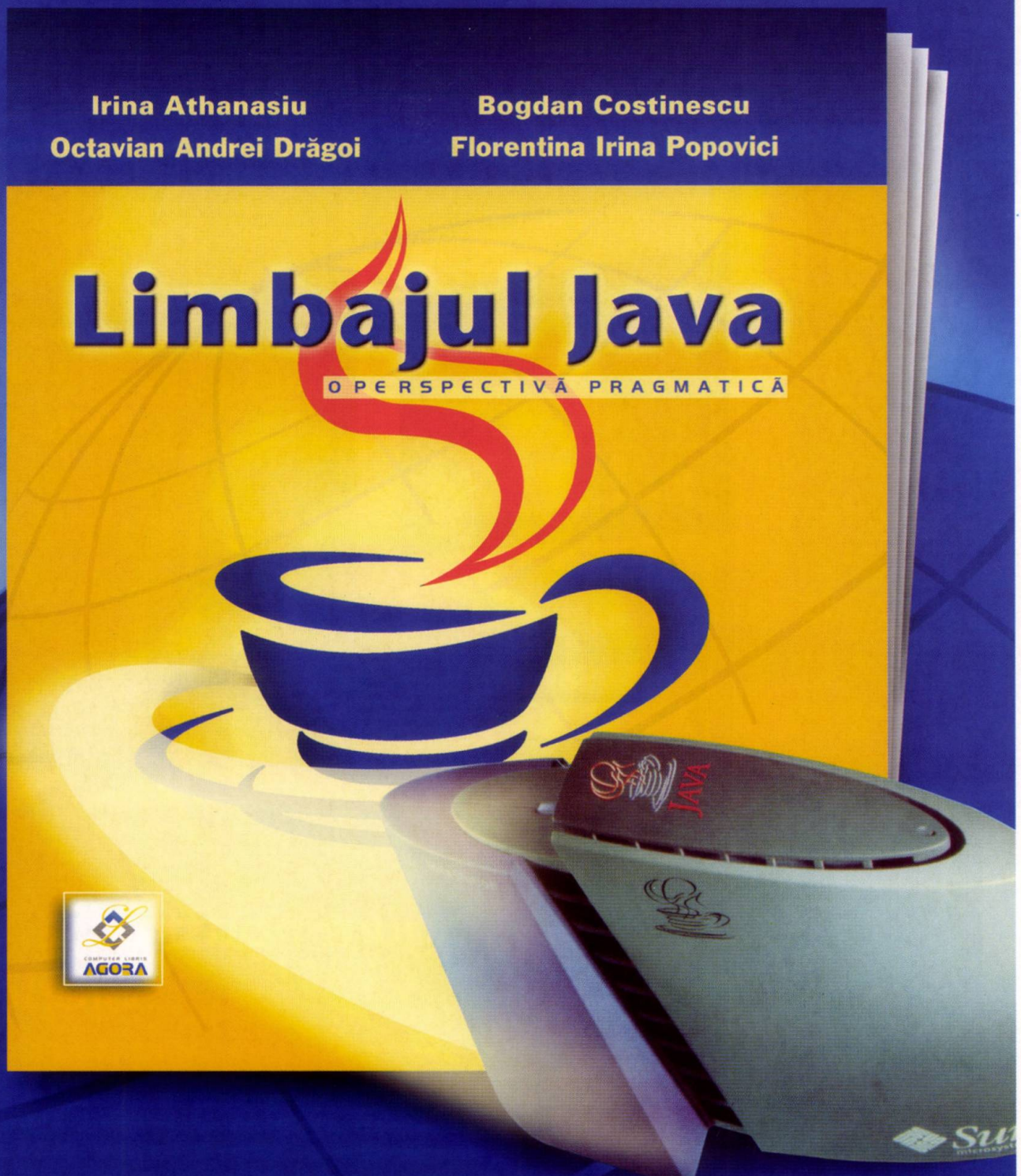
Premiul lunii

Abonatul desemnat prin tragere la sorți luna aceasta este **Sorin Groza** din Aiud, care va primi un premiu constând din cărțile *Limbajul Java* de Eugen Rotariu și *Internet* de Arnaud Dufour (plus câteva CD-ROM-uri).

Mircea Sârbu

tea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Car

COMPUTER LIBRIS AGORA



COMPUTER LIBRIS AGORA Cluj-Napoca, Str.Universității Nr.8/8, 3400, Tel. 064-192422
COMPUTER PRESS AGORA Tg.Mureș Str.Tudor Vladimirescu Nr. 63/1 (CP 172/1), Tel. 065-166516, Fax 065-166290
Sucursala București: Str.Poterași, Nr.35/2 (CP 94/49), Tel: 01-3309281, Fax: 01-3309285

Ştiţi cât costă un IBM PC?

doar 1100 USD !!!

Dacă pe lângă cei **800** USD cât costă un pachet **LOTUS SmartSuite** mai plătiţi **300** USD, primiţi şi un calculator original **IBM PC300GL**, în următoarea configuraţie:

- procesor INTEL Pentium MMX 200MHz
- 16MB RAM
- 2.1GB HDD
- placă video PCI 1MB VRAM
- monitor IBM G42 14"
- tastatură IBM 104 taste UK
- mouse IBM
- Windows 95 preinstalat, cu licenţă
- licenţă LOTUS SmartSuite

şi nu este singura noastră ofertă !!!

IBM® Business Partner™

Microsoft®
WHERE DO YOU WANT TO GO TODAY?™
Authorized Distribution Partner/
DSP for Romania

FORTE COMPUTERS
Flexibility.Made in Romania

IBM® Business Partner

Lipscani 102, Bucureşti
Tel: 3122360/6137282/3120948 Fax: 3122630

Gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

Februarie 1999
Vol. 9/2

Lei 10.000

din sumarul numărului:

Branch&Bound si Backtracking

Numere prime și divizibilitate

TEHNO
Microprocesoare

PROGRAMEAZĂ
PLACA DE SUNET

BAC
Subiect
Programa

INTERNET
Internet
Criptografie

χριπτο

BEGIN
Aiud
InfoStar '98

din sumarul numărului:

Branch&Bound si Backtracking

COMPUTER PRESS AGORA

PC & BYTE REPORT

TEHNOLOGIA INFORMAȚIEI

ECHIPAMENTE • SISTEME • APLICAȚII • REȚELE • INTERNET

*Abonează-te și plătești doar 120.000 lei (cu 60.000 lei mai puțin decât prețul de vânzare), evitând și ulterioarele majorări de preț

COMPUTER PRESS AGORA • CP.230 OP.1 TG.MUREȘ, Tel. 065-166516 Fax 065-166290

Vol. 9, Nr. 2 1999

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Editor executiv:
Mircea Sârbu (Târgu-Mureș)
Au mai colaborat:
Mihai Stroe (București)
Mihai Oltean (Cugir)
Ovidiu Gheorghioiu (Alba Iulia)
Mihai Scorțaru (Cluj)
Adrian Monea (Cluj)
Victor-Valeriu Patriciu (București)
Sorin Groza (Aiud)

Tehnoredactare:
Dan Crețu, Călin Borz
Coperta și grafica:
Mircea Drăgoi

Adresa pentru corespondență:
C.P. 172-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: (065) 166.516
Fax: (065) 166.290
http://www.ginfo.ro

ISSN: 1222-7129

Editor: Computer Press Agora srl
str. T. Vladimirescu nr. 63/1
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director editorial:
Mircea Sârbu
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (mszabo@agora.ro)
Denis Mărioara (mdenis@agora.ro)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: (01) 330.92.81

Sumar

soluții.. PAG. 4

probleme REZOLVATE
Mihai STROE

Soluții date la IOI '98

probleme.. PAG. 13

propuse pentru REZOLVARE

***

Probleme propuse la faza regională a concursului ACM

bac.. PAG. 19

Bacalaureat 1999
Clara IONESCU

Programa de BAC și un subiect de "probă"

focus.. PAG. 26

Numere prime și divizibilitate
Mihai OLTEAN

Branch&Bound și backtracking
Ovidiu GHEORGHIOIU

tehno.. PAG. 34

Microprocesoare
Mihai SCORȚARU

Programarea plăcii de sunet
Adrian MONEA

internet.. PAG. 44

Criptografia în Internet
Victor-Valeriu PATRICIU

Secretele lumii și protecția datelor

serial.. PAG. 49

Interfețe în Java
Horia GEORGESCU

Pagini WEB cu HTML 4.0
Mihai SCORȚARU

begin.. PAG. 61

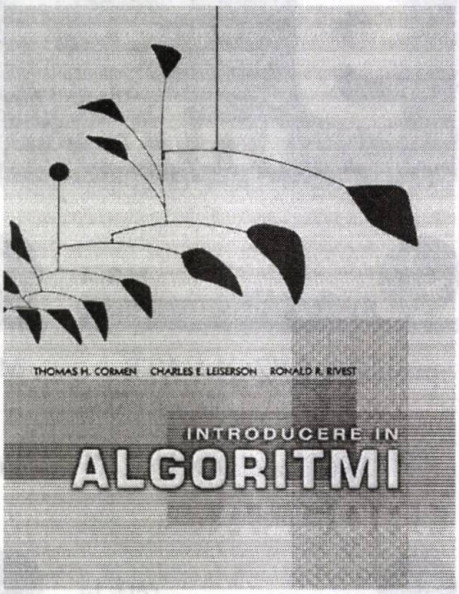
"Moș Nicolae" la Clubul Copiilor
Sorin GROZA

Programare la cluburile de copii

COMPUTER LIBRIS AGORA

În curând o nouă apariție editorială

La începutul lunii mai va apare în sfârșit și în limba română celebra carte de ALGORITMI scrisă de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, publicată de MIT Press, SUA. Este o carte care de la prima ei apariție a fost retipărită, numai în Statele Unite, de 21 ori.



Această carte oferă o introducere exhaustivă a studiului modern al algoritmilor. Analiza și proiectarea algoritmilor tratați este accesibilă și asigură condițiile unui studiu, pe de o parte fundamental, pe de altă parte profund.

Fiecare capitol prezintă cel puțin un algoritm sau o tehnică de proiectare, un domeniu de aplicații, respectiv un subiect înrudit. Algoritmii sunt descriși în limba română și în pseudocod. Cartea conține peste 260 de figuri care ilustrează modul de funcționare al algoritmilor. Deoarece accentul este pus pe eficiență ca și criteriu de proiectare, include analize atente ale timpului de executare a tuturor algoritmilor.

Cartea originală conține 1000 pagini și este structurată în șapte părți, subdivizate în 37 capitole, cu o bibliografie de 205 titluri și o listă bibliografică de 30 pagini.

Partea I, "Fundamente matematice", este un compendiu al metodelor și instrumentelor care vor fi utilizate în analiza algoritmilor. Este organizată esențial pentru referință, ca un fel de tutorial.

Partea a II-a, "Sortare și statistici de ordine", tratează problematica tehnicilor de sortare. Sunt studiate metodele *heapsort, quicksort și tehnici de sortare în timp liniar, precum și statisticile de ordine.*

Partea a III-a, "Structuri de date", prezintă unele tehnici de bază pentru reprezentarea și manipularea mulțimilor dinamice finite. Sunt discutate *structuri de date elementare, tabele de dispersare, arbori de căutare binară, arbori roșu-negru, extinderea structurilor de date.*

Partea a IV-a, "Tehnici avansate de proiectare și implementare", acoperă trei tehnici importante pentru proiectarea și analiza algoritmilor eficienți: *programarea dinamică, algoritmi greedy și analiza amortizată.* Aceste tehnici sunt ceva mai sofisticate decat altele, dar sunt esențiale pentru faptul că permit abordarea efectivă a multor probleme computaționale.

Partea a V-a, "Structuri de date avansate", revine la examinarea structurilor de date care suportă operații pe mulțimi dinamice, dar la un nivel mai avansat decat cel din Partea a III-a. Sunt discutați *B-arbori, ansambluri binomiale, ansambluri Fibonacci, precum și structuri de date pentru mulțimi disjuncte.*

Partea a VI-a, "Algoritmi de grafuri", tratează unele dintre cele mai importante probleme computaționale definite pe grafuri. Sunt trecuți în revistă algoritmi elementari pe grafuri, arbori de acoperire minimă, problematica celor mai scurte drumuri și algoritmi pentru determinarea fluxului maxim.

Partea a VII-a, "Subiecte diverse", conține o selecție de subiecte care extinde și completează materialul din restul cărții. Unele capitole introduc modele de calul noi precum circuite combinatoriale sau computere paralele. Altele acoperă domenii mai specializate precum geometria computațională sau teoria numerelor. Ultimele două capitole discută despre unele dintre limitările cunoscute ale proiectării algoritmilor eficienți și introduc tehnici pentru studierea acestor limitări.

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422]

Se apropie OLIMPIADELE

Lumea din jur este într-o schimbare permanentă, deci nu este de mirare că se schimbă și modul în care se evaluează lucrările concurenților, realizate cu ocazia olimpiadelor de informatică.

De câțiva ani, s-a introdus, în faza de evaluare a lucrărilor concurenților o metodă "modernă" de corectare, bazată pe ceea ce numim ***evaluare automată***. În ce constă acest stil de corectare? În primul rând este 100% fiabil. Nu mai pot interveni factori, cum ar fi subiectivismul, atenția, graba, sau chiar faptul că cel care a corectat "nu a văzut bine" rezultatul afișat pe ecran sau în fișier. Apoi, evaluarea automată este rapidă.

Fiind vorba de olimpiade de informatică, ce poate fi mai firesc, decât să se folosească instrumentul cel mai la îndemână: calculatorul. Dar cum nu există (încă?) nici un calculator pe lume care să poată *aprecia singur* dacă un program furnizează rezultate corecte sau nu, este nevoie de ***programe de verificare***. Aceste programe sunt scrise, de regulă, de autorul problemei, care cunoaște detalii cu privire la soluțiile posibile și poate estima eventualele greșeli.

Dacă soluțiile programelor corecte sunt unice, (de exemplu, când se cere determinarea unui singur număr - un *minim* sau un *maxim*), programul de verificare este simplu: rezultatul corect, așteptat și de la concurenți, există într-un fișier al comisiei; programul de verificare va compara conținutul acestuia cu cel al fișierelor concurenților. Dacă în urma verificării se constată că acestea diferă, concurentul nu primește puncte pentru testul respectiv.

Ce se întâmplă atunci când rezultatul nu este unic, de exemplu, când pe lângă minim se cer și termenii sumei care formează acel minim. Poate să difere ordinea, dar de multe ori pot să difere și termenii. În acest caz, programul de verificare, va calcula suma termenilor din rezultatul concurentului și va controla validitatea termenilor.

Și după cum bine știți, mai există un factor important în evaluare: *timpul de executare*. Anumite programe furnizează rezultatele într-un timp care se încadrează în limita precizată în specificațiile problemei, altele nu. Chiar mai mult, există programe care ar rula într-un timp inadmisibil de mare: ore sau zile. Cum, nu este posibil să se aștepte încheierea normală a executării, dar nici întreruperea manuală nu este cea mai potrivită soluție, este nevoie de un program care măsoară timpul de executare al programului concurentului și îl întrerupe după expirarea timpului.

Sunt cam multe programe... "Oare nu se încurcă membrii comisiei printre ele?" Evident, nu. Software-ul de evaluare controlează executarea tuturor acestor componente.

Evaluarea automată

Voi descrie în linii mari pașii pe care îi execută un asemenea soft, creat pentru a fi rulat sub DOS.

- Pentru a fi testate, programele (executabilele) concurenților trebuie să se afle într-un director, în care le va "căuta" softul de evaluare.
- Se copiază fișierele de intrare în acest director; aceste fișiere nu sunt denumite ca în specificațiile din enunț.
- Urmează o parte din algoritmul de evaluare care se repetă pentru fiecare test existent:
  - se redenumește fișierul de intrare corespunzător testului, astfel încât numele să fie cel din specificații;
  - se lansează programul care execută programul concurentului, măsoară timpul și îl întrerupe dacă acestuia îi expiră timpul;
  - se redenumește fișierul de ieșire, astfel încât numele acestuia să corespundă specificațiilor evaluatorului.
- Urmează partea de evaluare propriu-zisă:
  - se verifică datele din fișierele de ieșire ale concurentului și se acordă punctajul corespunzător fiecărui test în parte;
  - există probleme în cazul cărora se acordă punctaj parțial: de exemplu, un anumit număr de puncte pentru fiecare subpunct al problemei, dar se pot acorda punctaje parțiale pentru găsirea unor soluții "aproape optime".
- Un evaluator poate genera rapoarte pentru fiecare concurent, eliminându-se astfel posibilitățile de a greși în redactarea manuală a clasamentului.

Menționez că la concursurile internaționale timpul de executare al programului este măsurat de un alt calculator, cele două fiind legate în serie.

Problemele

Pentru a asigura o bună desfășurare pentru un asemenea concurs, evident, nu este suficient să se lucreze cu un evaluator automat. Probleme clar formulate, fără ambiguități, teste "acoperitoare", un punctaj bine gândit, o soluție corectă care *ar primi punctaj maxim în concurs* din partea autorului, toate contribuie la derularea fără incidente a concursurilor de informatică.

Clara Ionescu

Din GInfo 1:

SOLUȚIILE problemelor propuse

Mihai Stroe

Cum se poate obține "aur absolut", adică premiul I cu punctaj maxim la Olimpiada Internațională de Informatică? După performanța de vârf a lui Radu Lupșa din 1993 (Argentina), România se poate mândri din nou cu o asemenea medalie datorită lui Mihai Stroe din București. Mihai a reușit (în septembrie1998, în Portugalia) să demonstreze că forța lui intelectuală, puterea de analiză și sinteză sunt la același nivel cu talentul necesar pentru a urca în top-ul clasamentului. Îl felicităm!

P019901: Camelot

Trebuie observat faptul că în majoritatea cazurilor nu este avantajos ca regele să se deplaseze "pe jos", ci ar trebui să călătorească "călare".

Pentru a pregăti "terenul" în scopul mutării cavalerilor, vom calcula, pentru fiecare cavaler `i` costul deplasării lui din poziția în care se află inițial în oricare din celelalte poziții ale tablei. Aceste valori se păstrează în șirul de matrice `FC`.

Punctul de întâlnire nu este cunoscut, rezultă că vom presupune, pe rând, că oricare poziție `(x,y)` de pe tablă este un punct de întâlnire posibil. Pentru fiecare poziție posibilă de întâlnire (`(x,y)`, `x=1,2,...,8`, `y=1,2,...,8`) vom calcula (în matricele `FFIN`) numărul minim de pași în care un cavaler oarecare ar putea să ajungă din pozițiile de pe tablă în poziția `(x,y)` respectivă.

Considerând că atât cavalerii, cât și regele se deplasează pe cel mai scurt drum spre poziția `(x,y)`, se calculează numărul total de mutări și se determină acea poziție `(x,y)` pentru care acest număr este minim.

Revenim la observația referitoare la faptul că probabil este mai avantajos ca regele să se deplaseze împreună cu un cavaler. Presupunem, pe rând, pentru toate pozițiile `(i,j)` posibile că ar putea să fie loc de întâlnire între oricare cavaler și rege. Calculăm numărul total de mutări necesare, folosind valorile din matricele determinate anterior și îl comparăm cu optimul găsit în prima fază a algoritmului.

Implementarea nu necesită alte "artificii" de optimizare deoarece numărul posibilităților este relativ mic.

Listing: CAMELOT.PAS

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-}
{$Q-,R-,S+,T-,V+,X+}
{$M 65520,0,655360}
type ar=array[1..8,1..8] of Byte;
var dr,x,y,i,j,k,oldm,l,m,n,dk,opt:Longint;
    fi,fo:Text;
    s:string;
    rx,ry:Byte;
    cx,cy:array[1..63] of Byte;
    ffin:ar;
    fc:array[1..63] of ar;

procedure readdata;
begin
  Assign(fi,'CAMELOT.IN');
  Reset(fi);
  Readln(fi,s);
  Close(fi);
  rx:=Ord(s[1])-Ord('A')+1;
  ry:=Ord(s[2])-Ord('1')+1;
  Delete(s,1,2);
  while Length(s)>0 do
  begin
    Inc(n);
    cx[n]:=Ord(s[1])-Ord('A')+1;
    cy[n]:=Ord(s[2])-Ord('1')+1;
    Delete(s,1,2)
  end
end;
```

```
procedure fill(var a:ar;x,y,k:Integer);
begin
  if (x<1) or (y<1) or (x>8) or (y>8) then
Exit;
  if a[x,y]<=k then Exit;
  a[x,y]:=k;
  fill(a,x+2,y+1,k+1);
  fill(a,x+2,y-1,k+1);
  fill(a,x+1,y+2,k+1);
  fill(a,x+1,y-2,k+1);
  fill(a,x-1,y+2,k+1);
  fill(a,x-1,y-2,k+1);
  fill(a,x-2,y+1,k+1);
  fill(a,x-2,y-1,k+1)
end;

procedure fillfromhorses;
begin
  for i:=1 to n do
  begin
    Fillchar(fc[i],Sizeof(fc[i]),255);
    fill(fc[i],cx[i],cy[i],0)
  end
end;

function max(i,j:Longint):Longint;
begin
  if i>j then max:=i else max:=j
end;

procedure react;
begin
  if m<opt then opt:=m
end;

procedure try;
begin
  Fillchar(ffin,Sizeof(ffin),255);
  fill(ffin,x,y,0);
  m:=0;
  for i:=1 to n do m:=m+fc[i][x,y];
  dr:=max(Abs(rx-x),Abs(ry-y));
  m:=m+dr;
  i:=0;
  react;
  for i:=1 to 8 do
    for j:=1 to 8 do
      for k:=1 to n do
      begin
        oldm:=m;
        dk:=max(Abs(rx-i),Abs(ry-j));
        Dec(m,dr);
        Dec(m,fc[k][x,y]);
        Inc(m,fc[k][i,j]);
        Inc(m,dk);
        Inc(m,ffin[i,j]);
        react;
        Inc(m,dr);
        Inc(m,fc[k][x,y]);
        Dec(m,fc[k][i,j]);
        Dec(m,dk);
        Dec(m,ffin[i,j]);
        if m<>oldm then 1:=1
      end
end;

procedure solve;
begin
  fillfromhorses;
  opt:=MaxLongint;
  for x:=1 to 8 do
    for y:=1 to 8 do try;
  Assign(fo,'CAMELOT.OUT');
  Rewrite(fo);
  Writeln(fo,opt);
  Close(fo)
end;

Begin
  readdata;
  solve
End.
```

P019902: Contact

Singura dificultate o constituie mărimea fișierului de intrare, care nu poate fi citit și stocat în memorie, dar nici prelucrarea caracter cu caracter nu este posibilă.

Analizând problema, se observă că inițial ar trebui citite A caractere (lungimea minimă a unei secvențe posibilă de depistat ca semnal care se repetă). Ar urma prelucrarea acesteia respectiv contorizarea ei, apoi prelucrarea secvenței obținute prin adăugarea la secvența de lungime A a următorului caracter citit ș.a.m.d. până la prelucrarea secvenței de lungime B.

Datorită faptului că prelucrarea secvențelor după fiecare adăugare a unui caracter nou, prelucrarea subsecvențelor de lungime A, A+1, ..., B din care face parte acest ultim caracter citit, am preferat o citire general valabilă și nu am tratat separat cazul citirii unei secvențe având lungime mai mică decât A.

De asemenea, se observă că astfel, după prelucrarea secvenței de lungime B, citirea continuându-se, caracterul cel mai "bătrân" devine inutil și este eliminat.

Frecvențele sunt păstrate într-o matrice x de dimensiuni $12 \times 2^{12} = 12 \times 4096$. De fapt se folosesc valorile de la x[i,0] până la x[i,2^i-1] pentru i între A și B. Valoarea lui x[i,j] este egală cu frecvența unui șir de lungime i care este reprezentarea binară a numărului j.

Matricea x are elemente de tip Longint deoarece numărul maxim posibil pentru frecvența de apariție a unei secvențe de caractere are limita superioară 2.000.000.

Listing: CONTACT.PAS

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-}
{$R-,S+,T-,V+,X+}
{$M 16384,0,655360}
type ar=array[0..4095] of Longint;
var i,j,k,n,A,B:Longint;
    fi,fo:Text;
    x:array[1..12] of ^ar;
    top:array[1..20] of Longint;
    min,mp,max,aux:Longint;
    c:Char;
    l:Longint;
    m,p:Word;

procedure readdata;
begin
  Assign(fi,'CONTACT.IN'); Reset(fi);
  Readln(fi,A); Readln(fi,B);
  Readln(fi,n);
  for i:=A to B do
  begin
    New(x[i]);
    Fillchar(x[i]^,Sizeof(x[i]^),0)
  end
end;

procedure introdu;
begin
  for j:=1 to n do
    if x[k]^[i]=top[j] then Exit;
  if x[k]^[i]>min
  then
  begin
    top[mp]:=x[k]^[i];
    min:=MaxLongint;
    mp:=0;
    for j:=1 to n do
      if top[j]<min
      then begin min:=top[j]; mp:=j end
  end
end;

procedure readk;
begin
  l:=1;  Read(fi,c);
  while c<>'2' do { m este sirul de biti, }
  begin    { corespunzator caracterelor }
    m:=m shl 1;      { din secventa       }
    if c='1' then Inc(m);
    for k:=A to B do
      if l>=k
      then
      begin
        p:=m and(1 shl k-1);
        Inc(x[k]^[p]);{ cresterea frecventei }
      end;      { sirurilor                  }
    Read(fi,c);
    Inc(l);
    m:=m mod (1 shl b)
  end;
  for k:=A to B do
    for i:=0 to (1 shl k)-1 do introdu
end;          { se obtin frecventele minime }

procedure afis;
begin
  for i:=1 to n do
  begin
    max:=top[i]; mp:=i;
    for j:=i+1 to n do
      if top[j]>max
      then begin max:=top[j]; mp:=j end;
    top[mp]:=top[i];
    top[i]:=max
  end;
  Assign(fo,'CONTACT.OUT'); Rewrite(fo);
  for i:=1 to n do
    if top[i]>0
    then
    begin
      Write(fo,top[i]);
      for k:=B downto A do
        for j:=1 shl k-1 downto 0 do
          if x[k]^[j]=top[i]
          then
          begin
            Write(fo,' ');
            l:=1 shl k;
            for p:=1 to k do
            begin
              l:=l shr 1;
              if j and l=0 then Write(fo,0)
                           else Write(fo,1)
            end
          end;
      Writeln(fo)
    end;
  Close(fo)
end;

procedure solve;
begin
  for i:=1 to n do top[i]:=-1;
  min:=-1; mp:=1;
  readk; Close(fi);
  afis
end;

Begin
  readdata;
  solve
End.
```

P019903: Orgă de lumini

Se observă că nu contează ordinea operațiilor și că apăsarea unui buton de N ori are același efect cu apăsarea lui de (N **mod** 2) ori. Astfel, din orice secvență se obține una simplificată prin cel mult patru apăsări de butoane diferite. Există 2^4=16 astfel de secvențe simplificate. Pentru a obține o secvență validă cu C apăsări este necesar să existe o secvență simplificată cu numărul de apăsări M de aceeași paritate cu C. Secvența de C apăsări se obține apăsând butonul 1 de încă M-C ori.

Programul generează cele 16 secvențe și testează validitatea lor.

O secvență validă are numărul de acționări mai mic sau egal cu C, de aceeași paritate cu C, iar rezultatul aplicării ei respectă restricțiile precizate prin datele de intrare.

Programul este foarte simplu: se citesc datele de intrare, se generează cele 16 secvențe simplificate, iar procedura TRY verifică dacă secvența curentă este bună. În caz afirmativ, se scrie în fișier rezultatul aplicării ei.

Listing: ORGA.PAS

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-}
{$R-,S+,T-,V+,X+}
{$M 16384,0,655360}
var i,j,k,l,m,n,c:Longint;
    fi,fo:Text;
    apr,st:set of Byte;
    b:array[1..4] of Byte;
    a:array[1..100] of Byte;

procedure readdata;
begin
  Assign(fi,'PARTY.IN'); Reset(fi);
  Readln(fi,n); Readln(fi,c);
  repeat
    Read(fi,i);
    if i<>-1 then apr:=apr+[i]
  until i=-1;
  Readln(fi);
  repeat
    Read(fi,i);
    if i<>-1 then st:=st+[i]
  until i=-1;
  Close(fi)
end;

procedure try;
begin
  m:=b[1]+b[2]+b[3]+b[4];
  if m>c then Exit;
  if (m mod 2)<>(c mod 2) then Exit;
  if b[1]=1 then Fillchar(a,Sizeof(a),0)
            else Fillchar(a,Sizeof(a),1);
  if b[2]=1 then
    for i:=1 to n do
      if i mod 2=1 then a[i]:=1-a[i];
  if b[3]=1 then
    for i:=1 to n do
      if i mod 2=0 then a[i]:=1-a[i];
  if b[4]=1 then
    for i:=1 to n do
      if i mod 3=1 then a[i]:=1-a[i];
  for i:=1 to n do
    if ((i in apr) and (a[i]=0)) or ((i in st)
                and (a[i]=1)) then Exit;
  for i:=1 to n do Write(fo,a[i]);
  Writeln(fo)
end;

procedure back(k:Longint);
begin
  if k=5 then begin try; Exit end;
  b[k]:=0;
  back(k+1);
  b[k]:=1;
  back(k+1)
end;

procedure solve;
begin
  Assign(fo,'PARTY.OUT'); Rewrite(fo);
  back(1);
  Close(fo)
end;

Begin
  readdata;
  solve
End.
```

P019904: Dreptunghiuri

Pentru siguranța că programul va rula în timp pe orice test, se impune complexitatea maximă n^2 și o implementare fără prea multe operații.

Numărul cerut este suma lungimilor unor segmente orizontale și verticale. Am elaborat un algoritm $O(n^2)$ pentru calcularea sumei lungimilor segmentelor paralele cu *Ox*, care se aplică din nou după o transformare a figurii pentru celelalte segmente (paralele cu *Oy*).

Procedurile FINDX și FINDY găsesc valorile pentru x și y care corespund coordonatelor vârfurilor unor dreptunghiuri. Prin punctele x se duc paralele la *Oy*. Procedura SORTPOLY ordonează dreptunghiurile după valoarea coordonatei verticale a colțului stânga jos.

Pentru fiecare două paralele consecutive la *Oy* se determină suma lungimilor segmentelor care aparțin intersecției conturului figurii cu zona din plan mărginită de cele două paralele. Algoritmul de determinare a acestei sume are ordinul de complexitate $O(n)$.

Se parcurg dreptunghiurile (în prealabil sortate) și pentru fiecare dreptunghi "activ" (care intersectează zona

respectivă), se actualizează "intervalul curent". La început limitele intervalului vor fi determinate de marginea superioară și marginea inferioară a primului dreptunghi găsit. Dacă un dreptunghi activ este cuprins în "intervalul curent", acest interval va fi extins până la limitele noului dreptunghi. Dacă dreptunghiul activ nu este cuprins în "intervalul curent", vom deschide un nou interval.

Ceea ce ne interesează este de fapt numărul intervalelor. Acesta, înmulțit cu dublul distanței dintre cele două paralele la *Oy*, reprezintă suma lungimilor segmentelor din zona respectivă.

Listing: DREPTUNGHI.PAS

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-}
{$R-,S+,T-,V+,X+}
{$M 16384,0,655360}
type drept=record
                x1,y1,x2,y2:Integer
              end;
     ar=array[1..5000] of drept;
     arr=array[-10000..10000] of Integer;
var p:Longint;
    y1,y2:array[1..5000] of Integer;
    i,j,k,l,m,n,nrx,nry,nrint:Integer;
    fi,fo:Text;
    x,y:array[1..10000] of Integer;
    dr:^ar;
    v:^arr;

procedure readdata;
begin
  New(dr);
  Fillchar(dr^,Sizeof(dr^),0);
  Assign(fi,'PICTURE.IN'); Reset(fi);
  Readln(fi,n);
  for i:=1 to n do
  begin
    Readln fi,dr^[i].x1,dr^[i].y1,dr^[i].x2,
                                  dr^[i].y2);
    if dr^[i].x1>dr^[i].x2
    then
    begin
      k:=dr^[i].x1;
      dr^[i].x1:=dr^[i].x2;
      dr^[i].x2:=k
    end;
    if dr^[i].y1>dr^[i].y2
    then
    begin
      k:=dr^[i].y1;
      dr^[i].y1:=dr^[i].y2;
      dr^[i].y2:=k
    end
  end;
  Close(fi)
end;
```

```
procedure findx;
begin
  Fillchar(v^,Sizeof(v^),0);
  for i:=1 to n do
  begin
    v^[dr^[i].x1]:=1;
    v^[dr^[i].x2]:=1
  end;
  nrx:=0;
  for i:=-10000 to 10000 do
    if v^[i]=1
    then begin Inc(nrx); x[nrx]:=i end
end;

procedure findy;
begin
  Fillchar(v^,Sizeof(v^),0);
  for i:=1 to n do
  begin
    v^[dr^[i].y1]:=1;
    v^[dr^[i].y2]:=1
  end;
  nry:=0;
  for i:=-10000 to 10000 do
  if v^[i]=1
  then begin Inc(nry); y[nry]:=i end
end;

procedure Sort(l,r:Integer);
var i,j,x:Integer;
    y:drept;
begin
  i:=l; j:=r; x:=dr^[(l+r) div 2].y1;
  repeat
    while dr^[i].y1<x do i:=i+1;
    while x<dr^[j].y1 do j:=j-1;
    if i<=j
    then
    begin
      y:=dr^[i]; dr^[i]:=dr^[j];
      dr^[j]:=y; i:=i+1; j:=j-1
    end
  until i>j;
  if l<j then Sort(l,j);
  if i<r then Sort(i,r)
end;

procedure sortpoly;
begin
  sort(1,n);
  for i:=1 to n do
  begin
    y1[i]:=dr^[i].y1;
    y2[i]:=dr^[i].y2;
  end
end;
```

```
procedure buildinterval;
begin
  nrint:=0;
  j:=0;
  l:=-30000;
  for i:=1 to n do
    if dr^[i].x1<=x[k]
    then
      if dr^[i].x2>x[k]
      then
      begin
        if (l=-30000) or (y1[i]>l)
        then begin Inc(nrint); l:=y2[i] end
        else
          if y2[i]>l then l:=y2[i]
      end;
  p:=p+2*nrint*(x[k+1]-x[k])
end;

procedure rotate;
begin
  for i:=1 to n do
  begin
    l:=dr^[i].x1;
    dr^[i].x1:=dr^[i].y1;
    dr^[i].y1:=l;
    l:=dr^[i].x2;
    dr^[i].x2:=dr^[i].y2;
    dr^[i].y2:=l
  end
end;

procedure solve;
begin
  New(v);
  Inc(nry);
  findx;
  findy;
  p:=0;
  sortpoly;
  for k:=1 to nrx-1 do buildinterval;
  rotate;
  findx;
  findy;
  sortpoly;
  for k:=1 to nrx-1 do buildinterval;
  Assign(fo,'PICTURE.OUT');
  Rewrite(fo);
  Writeln(fo,p);
  Close(fo)
end;
Begin
  readdata;
  solve
End.
```

P019905: Poligon

Problema este similară altor probleme în care se cere parantezarea unei expresii și se rezolvă prin metoda programării dinamice.

Pentru fiecare eliminare a unei laturi, obținem o expresie aritmetică, în care întâlnim numere întregi și operatorii + și *. Această expresie trebuie parantezată astfel încât rezultatul obținut să fie maxim.

În matricele `MIN` și `MAX` se calculează valorile minime, respectiv maxime pentru subexpresii ale expresiei inițiale; `MIN[i,j]`, de exemplu, reprezintă minimul subexpresiei cu operanzii de la `i` la `j` din expresia inițială.

Valorile care se cunosc inițial sunt `MIN[i,i]` și `MAX[i,i]`, care sunt chiar operanzii respectivi. Rezultatul este dat de `MAX[1,n]`.

Procedura `INIT` generează, pe rând, expresiile cărora li se va calcula maximul.

Minimul și maximul pentru o expresie se obțin din subexpresiile ei. Să presupunem că am ales cele două subexpresii. Dacă între ele este semnul "+", `MIN[i,j]` și `MAX[i,j]` se obțin prin suma minimelor, respectiv maximelor subexpresiilor. Pentru înmulțire, apar mai multe cazuri; se justifică astfel păstrarea valorii pentru `MIN` (de exemplu, un maxim pozitiv se poate obține înmulțind două minime negative, dar mari în modul).

Listing: POLIGON.PAS

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-}
{$R-,S+,T-,V+,X+}
{$M 16384,0,655360}
var i,j,k,l,m,n,lung:Longint;
    fi,fo:Text;
    nr,a:array[1..100] of Integer;
    v,s:array[1..100] of Char;
    opt:Longint; c:Char;
    sol:set of Byte;
    min,max:array[1..50,1..50] of Longint;

procedure readdata;
begin
  Assign(fi,'POLYGON.IN'); Reset(fi);
  Readln(fi,n);
  for i:=1 to n do Read(fi,v[i],nr[i],c);
  Close(fi);
  for i:=1 to n do
  begin v[n+i]:=v[i]; nr[n+i]:=nr[i] end
end;

procedure init;
begin
  for i:=1 to n do
  begin
    a[i]:=nr[i+k-1];
    if i<>n then s[i]:=v[i+k]
  end
end;
```

```
procedure try;
begin
  for i:=1 to n do
  begin min[i,i]:=a[i]; max[i,i]:=a[i] end;
  for lung:=2 to n do
    for i:=1 to n-lung+1 do
    begin
      j:=i+lung-1;
      min[i,j]:=100000;
      max[i,j]:=-100000;
      for l:=i to j-1 do
        case s[l] of
          't': begin
               if min[i,l]+min[l+1,j]<min[i,j]
               then
                 min[i,j]:=min[i,l]+min[l+1,j];
               if max[i,l]+max[l+1,j]>max[i,j]
               then
                 max[i,j]:=max[i,l]+max[l+1,j]
               end;
          'x': begin
               if min[i,l]*min[l+1,j]>max[i,j]
               then
                 max[i,j]:=min[i,l]*min[l+1,j];
               if max[i,l]*max[l+1,j]>max[i,j]
               then
                 max[i,j]:=max[i,l]*max[l+1,j];
               if min[i,l]*max[l+1,j]>max[i,j]
               then
                 max[i,j]:=min[i,l]*max[l+1,j];
               if max[i,l]*min[l+1,j]>max[i,j]
               then
                 max[i,j]:=max[i,l]*min[l+1,j];
               if min[i,l]*min[l+1,j]<min[i,j]
               then
                 min[i,j]:=min[i,l]*min[l+1,j];
               if max[i,l]*max[l+1,j]<min[i,j]
               then
                 min[i,j]:=max[i,l]*max[l+1,j];
               if min[i,l]*max[l+1,j]<min[i,j]
               then
                 min[i,j]:=min[i,l]*max[l+1,j];
               if max[i,l]*min[l+1,j]<min[i,j]
               then
                 min[i,j]:=max[i,l]*min[l+1,j]
               end
        end
    end;
  if max[1,n]=opt
  then sol:=sol+[k]
  else
    if max[1,n]>opt
    then begin
           opt:=max[1,n]; sol:=[k]
         end
end;
```

```
procedure afis;
begin
  Assign(fo,'POLYGON.OUT');
  Rewrite(fo);
  Writeln(fo,opt);
  l:=0;
  for i:=1 to n do
    if i in sol
    then
    begin
      if l=0
      then Write(fo,i)
      else Write(fo,' ',i);
      Inc(l)
    end;
  Writeln(fo);
  Close(fo)
end;

procedure solve;
begin
  opt:=-100000;
  for k:=1 to n do
  begin
    init; try
  end;
  afis
end;

Begin
  readdata;
  solve
End.
```

P019906: Cer înstelat

Grupările de stele se memorează ca liste de puncte în care fiecărei stele i se asociază coordonate relative astfel încât coordonatele minime vor fi `x=1` și `y=1`. În cadrul listei, stelele sunt ordonate după linii, iar în cadrul aceleiași linii după coloane. *Testarea egalității a două liste devine astfel liniară.* O grupare se determină cu ajutorul procedurii `FILL`, introducând steaua curentă în listă, marcând-o pentru a nu fi vizitată a două oară și apelând procedura `FILL` pentru pozițiile vecine. Procedurile pentru translatare, ordonare, rotație și oglindire sunt rapide și relativ ușor de înțeles.

Practic, fiecare grupare nouă este căutată în lista "claselor" de grupări. Apoi se caută și celelalte 7 reprezentări posibile (obținute prin rotații și/sau oglindiri). În cazul în care o grupare nu este regăsită, ea este memorată în listă; în caz contrar, gruparea se marchează cu litera care îi revine.

S-a ales reprezentarea cu liste de stele (în locul celei cu matrice) pentru a obține un timp mai bun (de exemplu, cu 101 stele se poate obține o matrice 50*50, mult mai mare decât o listă, deci egalitatea a două grupări, rotația și oglindirea durează mai mult).

Listing CER.PAS

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-}
{$R-,S+,T-,V+,X+}
{$M 65520,0,655360}
type comp=record
            x,y:array[1..160+5] of Integer;
            nr:Word
          end;
var i,j,k,l,m,n,ii,jj,x1,y1:Longint;
    fi,fo:Text;
    a:array[1..100] of string;
    sol:array[1..27] of comp;
    c:comp;ch:Char;
    gata:Boolean;
    minx,miny,maxx,maxy:Longint;

procedure readdata;
begin
  Assign(fi,'STARRY.IN');
  Reset(fi);
  Readln(fi,n);
  Readln(fi,m);
  for i:=1 to m do
    Readln(fi,a[i]);
  Close(fi)
end;

procedure afis;
begin
  Assign(fo,'STARRY.OUT');
  Rewrite(fo);
  for i:=1 to m do
    Writeln(fo,a[i]);
  Close(fo)
end;

procedure fill(i,j:Byte);
begin
  if (i=0) or (j=0) or (i>m) or (j>n)
  then Exit;
  if a[i,j]<>'1'
  then Exit;
  a[i,j]:=ch;
  Inc(c.nr);
  c.x[c.nr]:=i;
  c.y[c.nr]:=j;
  fill(i-1,j-1);
  fill(i-1,j);
  fill(i-1,j+1);
  fill(i,j-1);
  fill(i,j+1);
  fill(i+1,j-1);
  fill(i+1,j);
  fill(i+1,j+1)
end;
```

```
function equals:Boolean;
begin
  equals:=false;
  if c.nr<>sol[l].nr
  then Exit;
  for i:=1 to c.nr do
    if (c.x[i]<>sol[l].x[i]) or
       (c.y[i]<>sol[l].y[i]) then Exit;
  equals:=true
end;

procedure Sort(l,r:Longint);
var i,j,x,y:Longint;
begin
  i:=l; j:=r;
  x:=c.x[(l+r) div 2]*200+c.y[(l+r)div 2];
  repeat
    while c.x[i]*200+c.y[i]<x do i:=i+1;
    while x<c.x[j]*200+c.y[j] do j:=j-1;
    if i<=j
    then
    begin
      y:=c.x[i];
      c.x[i]:=c.x[j];
      c.x[j]:=y;
      y:=c.y[i];
      c.y[i]:=c.y[j];
      c.y[j]:=y;
      i:=i+1;
      j:=j-1
    end
  until i>j;
  if l<j then Sort(l,j);
  if i<r then Sort(i,r)
end;

procedure checkcomp;  { sortare dupa x si y }
begin
  sort(1,c.nr);
  for l:=1 to k do
    if equals
    then
    begin
      gata:=true;
      for i:=1 to m do
        for j:=1 to n do
          if a[i,j]=ch
          then a[i,j]:=Chr(Ord('a')+l-1);
      Exit
    end
end;

procedure scalecomp;
begin
  minx:=c.x[1];
  miny:=c.y[1];
```

```
  for i:=2 to c.nr do
  begin
    if c.x[i]<minx
    then minx:=c.x[i];
    if c.y[i]<miny
    then miny:=c.y[i]
  end;
  for i:=1 to c.nr do
  begin
    c.x[i]:=c.x[i]-minx+1;
    c.y[i]:=c.y[i]-miny+1
  end;
end;

procedure rotatecomp;
begin
  maxx:=c.x[1];
  maxy:=c.y[1];
  for i:=2 to c.nr do
  begin
    if c.x[i]>maxx
    then maxx:=c.x[i];
    if c.y[i]>maxy
    then maxy:=c.y[i]
  end;
  for i:=1 to c.nr do
  begin
    y1:=c.y[i];
    c.y[i]:=maxx-c.x[i]+1;
    c.x[i]:=y1
  end
end;

procedure reflectcomp;
begin
  maxx:=c.x[1];
  for i:=2 to c.nr do
    if c.x[i]>maxx
    then maxx:=c.x[i];
  for i:=1 to c.nr do
    c.x[i]:=maxx-c.x[i]+1
end;

procedure try;
begin
  gata:=false;
{1}
  scalecomp;
  checkcomp;
  if gata then Exit;
{2}
  rotatecomp;
  scalecomp;
  checkcomp;
  if gata then Exit;
{3}
  rotatecomp;
  scalecomp;
  checkcomp;
  if gata then Exit;
{4}
  rotatecomp;
  scalecomp;
  checkcomp;
  if gata then Exit;
{5}
  reflectcomp;
  scalecomp;
  checkcomp;
  if gata then Exit;
{6}
  rotatecomp;
  scalecomp;
  checkcomp;
  if gata then Exit;
{7}
  rotatecomp;
  scalecomp;
  checkcomp;
  if gata then Exit;
{8}
  rotatecomp;
  scalecomp;
  checkcomp;
  if gata then Exit;
  Inc(k);
  sol[k]:=c;
end;

procedure solve;
begin
  k:=0;
  for ii:=1 to m do
    for jj:=1 to n do
      if a[ii][jj]='1'
      then
      begin
        ch:=Chr(Ord('a')+k);
        Fillchar(c,Sizeof(c),0);
        fill(ii,jj); try
      end;
  afis
end;

Begin
  readdata;
  solve
End.
```

Mihai Stroe este student la Universitatea Politehnică din București; poate fi contactat prin e-mail la adresa: mstroe@pcnet.pcnet.ro

Încercați-vă puterile!

PROBLEME propuse pentru REZOLVARE

În toamna anului trecut a avut loc etapa regională a concursului studențesc ACM, la care au participat 35 echipe din mai multe țări. Concursul a fost câștigat de prima echipă a Universității București, urmată de prima echipă a Universității Politehnice din București. Aceste două echipe vor participa la etapa finală a concursului ACM, organizată anul acesta la Eindhoven (Olanda).

P029901: Țara Confuziei

În călătoria sa în jurul lumii un turist a ajuns în Țara Confuziei. În această țară există mai multe intersecții unite prin străzi cu sens unic. La fiecare intersecție pot exista mai multe străzi marcate cu "Nord", "Est", "Sud" sau "Vest". Când turistul ajunge la o intersecție el alege una dintre străzile marcate în planul său de călătorie și se îndreaptă spre intersecția la care duce această stradă. Deoarece semnele sunt ambigue sau nepotrivite pot apărea câteva situații ciudate:

- Nu sunt marcate decât străzile care pornesc dintr-o intersecție, nu și cele care sosesc în intersecție. Turistul poate pleca dintr-o intersecție numai pe o stradă marcată (cu sens unic).
- Pentru fiecare semn (N,E,V,S) pot exista mai multe străzi care pleacă în acea direcție, dar se poate întâmpla să nu existe nici una.
- Dacă turistul vine din intersecția *A* și ajunge în intersecția *B* printr-o stradă marcată cu "Nord" în intersecția *A*, nu înseamnă neapărat că el poate ajunge înapoi din *B* în *A* alegând o stradă marcată cu "Sud"; poate exista o stradă de la *B* la *A* marcată cu un alt semn sau s-ar putea să nu existe nici o stradă de la *B* la *A*.
- O stradă marcată, de la intersecția *A* poate duce la aceeași intersecție.

Tabelul 1

Intersecția	Intersecțiile la care ajung străzile marcate			
	Nord	Est	Sud	Vest
1	2, 3		1	
2		4		3
3	5	2, 4		
4				5
5	6		1	
6				

De exemplu, tabelul 1 definește o hartă a străzilor în Țara Confuziei.

În acest exemplu călătorul aflat în intersecția 1 poate alege una dintre străzile spre Nord pentru a ajunge fie în intersecția 2, fie în intersecția 3, sau să meargă spre Sud și să ajungă tot în intersecția 1. Turistul nu are alte opțiuni atunci când se află în intersecția 1. Din intersecția 5 el poate merge fie spre Nord, și ajunge astfel în intersecția 6, fie spre Sud, și ajunge în intersecția 1.

Turistul are mai multe planuri de călătorie. Fiecare plan este o secvență formată din literele 'N', 'E', 'S' și 'V' care indică direcția pe care turistul ar trebui să o aleagă în intersecția în care se află. Un plan este considerat corect dacă există un drum de lungime egală cu lungimea secvenței între intersecția de pornire și intersecția de sosire, drum care trebuie să respecte direcțiile precizate în plan. În caz contrar planul este incorect. Observați că pentru același plan pot exista nici unul, unul sau mai multe astfel de drumuri.

Pentru exemplul dat, dacă turistul pornește din intersecția 1 și dorește să ajungă în intersecția 6, atunci secvența 'SSSNEEVN' este corectă deoarece există drumul: 1 (plecare), 1, 1, 1, 3, 2, 4, 5, 6 (sosire) de lungime 8 care duce la sosire. Secvența 'NEVNS' este, în acest caz, incorectă: turistul poate alege drumul: 1 (plecare), 2, 4, 5, 6 și apoi nu poate merge spre Sud. Observați că acest drum nu este corect, deoarece nu are aceeași lungime ca și secvența. Turistul mai poate alege drumul: 1 (plecare), 3, 2, 3, 5, 1, (ultima intersecție nu este sosirea dorită) sau drumul: 1 (start), 3, 4, 5, 6 (lungimea drumului nu este aceeași cu lungimea secvenței).

Dându-se o hartă a străzilor, scrieți un program care va afla care dintre planurile date sunt corecte și care nu, precum și intersecția de plecare și cea de sosire.

Fișierul de intrare conține mai multe seturi de date. Fiecare set de date reprezintă o hartă de străzi, intersecția de plecare și cea de sosire, precum și câteva planuri date. Există exact o linie vidă între două seturi de date consecutive.

Fiecare set de date începe cu o linie care conține un întreg care reprezintă numărul de intersecții. Numărul 0 marchează sfârșitul fișierului de intrare. Intersecțiile sunt numerotate pornind de la 1. Pe linia următoare se află doi întregi separați prin spații care indică intersecția de plecare și cea de sosire. Următoarele linii sunt grupate câte patru pentru fiecare intersecție. Aceste patru linii indică intersecțiile adiacente (separate prin spații) în care se poate ajunge direct din intersecția dată prin străzile marcate cu Nord (prima linie), Est (a doua linie), Sud (a treia linie) și Vest (a patra linie). Dacă o linie conține un singur 0 înseamnă că nu există străzi marcate cu semnul respectiv în intersecția dată. Urmează o linie care conține un singur întreg NP care indică numărul planurilor date. Următoarele NP linii conțin secvențele (o secvență pe linie). Planul constă din literele 'N', 'E', 'S' sau 'V'. Numărul maxim de intersecții este de 30, și lungimea maximă a planului este 80.

Pentru fiecare set de date, programul trebuie să afișeze o linie care să conțină propoziția 'Fuzzyland #:' și câte o linie pentru fiecare plan care să cuprindă răspunsul 'DA' dacă secvența este corectă sau 'NU' în caz contrar.

Exemplu:

Intrare: **A.DAT**

```
6       5       0       5 6
1 6     6       6       0
2 3     0       0       7
0       1       0       0
1       0       3       2
0       0       0       0
0       0       5       0
4       0       2       0
0       0       0       0
3       3       0       0
5       SSSNEEVN  4 5     3
2 4     NEVNS     6       NEVENSVSSN
0       SNEVSNEVN 0       SSVN
0                 0       SNN
0       7         0
0       1 7       0       0
0       2         4
```

Ieșire: - ecran

```
Fuzzyland #1:
DA
NU
DA

Fuzzyland #2:
DA
NU
NU
```

P029902: Numărul majoritar

Numărul majoritar al unui șir nevid de N numere este acel număr care apare de mai mult de N/2 ori în secvență. De aceea, pentru un șir nevid, există cel mult un astfel de număr. De exemplu, șirul 3, 3, 4, 2, 4, 4, 2, 4, 4 are numărul majoritar 4, iar șirul 3, 3, 4, 2, 4, 4, 2, 4 nu are un număr majoritar.

Un fișier text conține șiruri nevide de întregi lungi. Fiecare șir, care poate fi destul de lung, începe cu un număr care indică numărul de întregi lungi din șir. Acest număr nu face parte din șir. Numerele pot fi separate prin caractere albe (spații, tab-uri și caractere de sfârșit de linie). Datele de intrare se consideră corecte.

Scrieți un program care, citește mai multe șiruri de întregi lungi din fișierul de intrare și afișează numărul majoritar al șirului. Dacă nu există un astfel de număr, se va scrie mesajul 'Nu exista numar majoritar'. Rezultatele se vor afișa la ieșirea standard pe linii separate.

Exemplu:

Intrare: **B.DAT**

```
6
100000 -2 56 100000 -2 56
5
-2 100 -2 -2 1001
1
1000
```

Ieșire - ecran

```
Nu exista numar majoritar
-2
1000
```

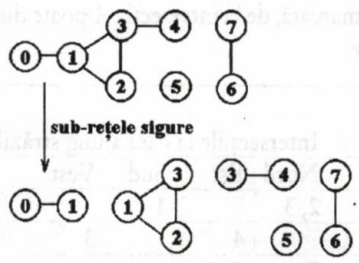
P029903: Rețele sigure

Un server *S* al unei rețele de calculatoare este considerat *critic* dacă există cel puțin alte două servere *A* și *B* astfel încât *A*, *B* și *S* sunt diferite și toate drumurile din rețea care conectează *A* și *B* trec prin *S* (o cădere a lui *S* deconectează *A* și *B*). Un exemplu este arătat în figură, unde serverele critice sunt 1 și 3. Vom spune că o rețea este *sigură* dacă nu conține servere critice. Se știe că:

(1) legăturile dintre servere sunt bidirecționale;

(2) un server nu poate fi conectat direct cu el însuși;

(3) rețeaua poate avea sub-rețele autonome.



Scrieți un program care găsește sub-rețelele sigure maximale dintr-o rețea dată. O *sub-rețea* este *maximală* dacă nu poate fi unită cu o altă sub-rețea sigură, astfel încât să

rezulte o rețea sigură mai mare. Programul trebuie să citească seturi de date dintr-un fișier text. Fiecare set specifică structura unei rețele și are următorul format:

```
nr_servere
server0(nr_conexiuni_directe)server_conectat...
                                server_conectat
                                ...
servernr_servere-1 (nr_conexiuni_directe)
                server_conectat ... server_conectat
```

Prima linie conține un întreg `nr_servere`≥0 care indică numărul de servere din rețea. Următoarele `nr_servere` linii (câte una pentru fiecare server din rețea), sunt ordonate în mod aleator și indică modul în care sunt conectate serverele. Linia corespunzătoare serverului $server_k$, $0 \le k \le nr\_servere-1$, specifică numărul conexiunilor directe dintre $server_k$ și serverele care sunt conectate direct cu acesta. Serverele sunt reprezentate prin întregi între `0` și `nr_servere-1`. Elementele unei linii de date sunt separate prin cel puțin un spațiu. Datele de intrare sunt corecte.

Rezultatul programului trebuie afișat la ieșirea standard. Pentru fiecare rețea din fișierul de intrare programul trebuie să afișeze numărul sub-rețelelor sigure maximale și apoi serverele fiecărei sub-rețele, câte o sub-rețea pe linie. Ieșirea pentru o rețea este urmată de o linie vidă. Serverele unei sub-rețele trebuie afișate în ordine crescătoare și separate printr-un singur spațiu. Sub-rețelele trebuie afișate în ordine crescătoare după prima lor componentă.

Exemplu:

Intrare: **C.DAT**

```
8
0 (1) 1
1 (3) 2 0 3
2 (2) 1 3
3 (3) 1 2 4
4 (1) 3
7 (1) 6
6 (1) 7
5 (0)
```

Ieșire: - ecran

```
5 sub-rețele
0 1
1 2 3
3 4
5
6 7
```

P029904: Drumuri pe autostradă

O companie dorește să construiască o autostradă între două orașe. În fazele inițiale ale proiectului compania folosește un satelit digital pentru a determina cele mai bune trasee pentru autostradă. Harta este reprezentată pe o grilă bidimensională ca o mulțime de puncte, închise într-un contur dreptunghiular. Conturul hărții și obstacolele de pe hartă sunt marcate folosind caracterul având codul ASCII 219. Punctele libere de pe hartă (pe unde poate trece autostrada) sunt marcate prin caracterele având codurile ASCII 176, 177 și 178. Orașele sunt marcate folosind caractere având codul ASCII 1. Există exact două orașe pe hartă. Marcajele pentru puncte libere, orașe și obstacole, pot apărea numai în interiorul conturului hărții; nu există alte marcaje pe hartă.

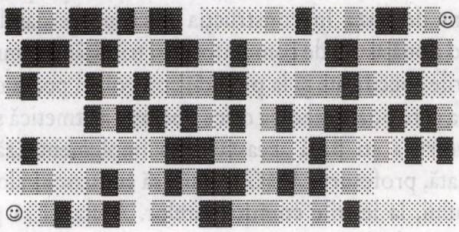
Un drum între două puncte de pe hartă unește (pe verticală și/sau orizontală) mai multe puncte libere adiacente. Un punct liber al cărui cod ASCII este `cod_punct` necesită (`cod_punct-175`) unități pentru construirea părții de autostradă ce trece prin acel punct. Costul asociat orașelor este `0`. Lungimea drumului este numărul punctelor sale, incluzând punctele corespunzătoare orașelor. Costul unui drum este suma dintre lungimea drumului și costul fiecărui punct de pe drum. Cel mai bun drum este drumul care are costul minim.

Scrieți un program care, pentru fiecare hartă citită dintr-un fișier text, găsește toate drumurile de cost minim dintre cele două orașe de pe hartă și marchează aceste drumuri folosind caracterul având codul ASCII 32 (spațiu). Toate drumurile trebuie marcate pe aceeași hartă. Orașele își mențin marcajul inițial.

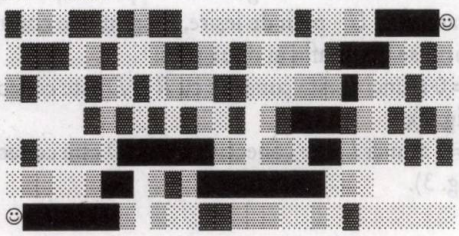
O hartă este citită ca o secvență de linii de text terminată printr-o linie care conține numai liniuțe de subliniere. Există cel mult 30 de linii, incluzând linia care indică terminarea hărții și cel mult 80 de caractere pe fiecare linie. Datele de intrare sunt corecte. Fișierul standard de ieșire conține hărțile cu drumurile marcate. Fiecare hartă este dată la ieșire în același format ca și la intrare, incluzând linia care indică terminarea hărții și este precedată de o linie care indică numărul celor mai bune drumuri și costurile lor (`0` dacă nu există drumuri). Numărul celor mai bune drumuri poate fi destul de mare și poate fi reprezentat pe patru cuvinte (8 bytes).

Exemplu:

Intrare: **D.DAT**

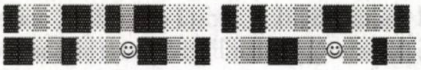


Ieșire: - ecran



Intrare: `D.DAT`

Ieșire - ecran



P029905: Inversiuni

Fie $a_1, a_2, ..., a_n$ un șir de `n` numere distincte. Dacă `i<j` și $a_i > a_j$ atunci perechea `(i,j)` se numește *inversiune*. De exemplu, în șirul 3, 2, 1, 5, 6, 7, 8, 9 există 3 inversiuni, în timp ce șirul 1, 2, 3, 4 nu are nici o inversiune.

Un fișier text conține șiruri nevide de întregi. Fiecare șir începe cu un număr `1≤N≤30000` care specifică numărul de întregi din șir. Acest număr nu face parte din șir. Numerele sunt separate prin caractere albe (spații, tab-uri și caractere de sfârșit de linie). Datele din fișierul de intrare sunt corecte.

Scrieți un program care, pentru fiecare șir de numere din fișierul de intrare, afișează numărul inversiunilor din șir. Fiecare rezultat trebuie afișat pe o linie separată, la ieșirea standard.

Exemplu

Intrare: `E.DAT`

```
5
1 2 3 4 5
4
4 3 2 1
1
1000
```

Ieșire: - ecran

```
0
6
0
```

P029906: Rețea de comutatoare

Profesorul *MAG* a inventat un comutator și l-a propus ca un element de bază în construcția microprocesoarelor. Conectând două astfel de comutatoare și trimițând semnale la intrările specificate și la pinii de control, el poate obține o rețea care poate rezolva orice operație aritmetică sau logică. Pentru a ști dacă această rețea implementează o operație dată, profesorul *MAG* trebuie să cunoască valorile semnalului, la ieșirile comutatoarelor. Ajutați-l să găsească aceste valori.

Comutatorul desenat în figura 1 are 5 pini: *L* și *D* - pinii de intrare, *R* și *U* - pinii de ieșire, *S* - pinul de control. Comutatorul se poate afla în două stări:

1. dacă *S* = 0 atunci *L* este conectat cu *R* și *D* este conectat cu *U* (fig. 2);
2. dacă *S* = 1 atunci *L* este conectat cu *U* și *D* este conectat cu *R* (fig. 3).

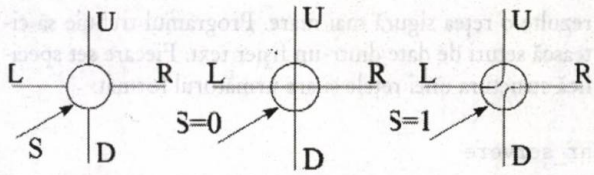


Fig. 1 Fig. 2 Fig. 3

Vom spune că două comutatoare sunt *conectate* dacă una dintre ieșirile primului comutator este conectată cu intrarea celui de-al doilea comutator. Rețeaua de comutatoare este o conexiune între elementele comutatoarelor.

Rețeaua de comutatoare, semnalele la unele intrări și semnalele tuturor pinilor de control sunt date. Vi se cere să găsiți semnalele la pinii de ieșire specificați, folosind informațiile date.

Intrare: - `F.DAT`

Structura fișierului de intrare este următoarea:

Pe prima linie este numărul seturilor de date. Următoarea linie începe cu primul set, iar seturile de date sunt separate printr-o linie vidă.

Setul de date are următoarea structură.

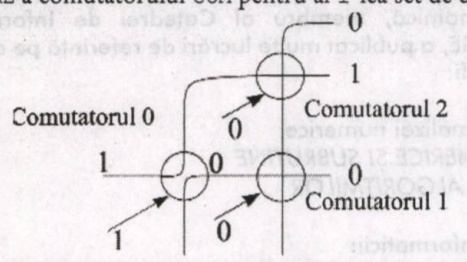
- Prima linie a setului de date conține numărul comutatoarelor `N`, unde `N≤4000` și numărul de conexiuni din rețea `K`.
- Comutatoarele sunt numerotate de la `0` la `N-1`. Intrările comutatoarelor au etichetele `L` și `D` iar ieșirile au etichetele `R` și `U`.
- Următoarele `K` linii descriu conexiunile dintre comutatoare. Fiecare linie are forma `COM1 IEȘIRE COM2 INTRARE` separate printr-un spațiu. `COM1` și `COM2` sunt numere care indică comutatoare, `IEȘIRE` este o ieșire a comutatorului `COM1`, `INTRARE` este o intrare a comutatorului `COM2`. Ieșirea `IEȘIRE` a comutatorului `COM1` este conectată cu intrarea `INTRARE` a comutatorului `COM2`. Numărul `COM2` este mai mare decât numărul `COM1`. Fiecare ieșire a unui comutator poate fi conectată numai la o intrare. Fiecare intrare a unui comutator poate fi conectată numai la o ieșire.
- Urmează o linie vidă.
- Următoarea linie conține `N` valori pentru semnalele pinilor de control ai comutatoarelor `0`, ..., `N-1`. Valorile posibile ale semnalului sunt `0` și `1`.
- Urmează o linie vidă.
- Putem trimite un semnal `1` la orice intrare care nu este conectată la o ieșire. Următoarea linie conține numărul `SG` de semnale cu valoarea `1` trimise intrărilor.
- Urmează `SG` linii de semnal. Fiecare linie de semnal conține numărul `COM` al comutatorului și intrarea `INTRARE`, separate printr-un spațiu. Înseamnă că semnalul cu valoarea `1` este trimis la intrarea `INTRARE` a comutatorului `COM`. Dacă intrarea unui comutator nu este conectată cu nici o ieșire și nu a fost trimis un semnal cu valoarea `1` la această intrare vom considera că acest pin este inutil (semnalul la această intrare este 0).
- Urmează o linie vidă.

Aceasta a fost descrierea structurii rețelei. Acum dorim să știm semnalele la pinii de ieșire specificați. Următoarea linie conține numărul de teste NT pentru setul de date curent.

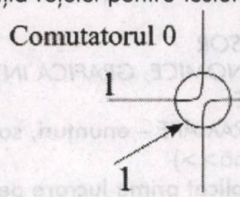
Următoarele NT linii de test au următoarea formă: numărul comutatorului COM și ieșirea IEȘIRE, separate printr-un singur spațiu. Pentru fiecare linie de test trebuie să aflăm semnalul la ieșirea IEȘIRE a comutatorului COM.

Ieșire: - ecran

Ieșirea constă din NPR linii, unde NPR este numărul seturilor de date. Fiecare linie conține NT semnale. Al i-lea număr în linia de ieșire curentă este un semnal la ieșirea IEȘIRE a comutatorului COM pentru al i-lea set de date.



Configurația rețelei pentru testul 1 din exemplu



Configurația rețelei pentru testul 2 din exemplu

Exemplu:

Figurile următoare prezintă două posibile rețele. Intrările și ieșirile corespunzătoare sunt prezentate mai jos.

Intrare: F.DAT

```
2
3 3
0 R 1 L
0 U 2 L
1 U 2 D

1 0 0

1
0 L

4
0 R
2 R
2 U
1 R

1 0

1

1
0 L

2
0 R
0 U
```

Ieșire: - ecran

```
0 1 0 0
0 1
```

P029907: Combinații

O *n-mulțime* este o mulțime cu *n* elemente. O *k-combinație* a unei *n-mulțimi S* este o *k*-submulțime a lui *S*. De exemplu 3-combinațiile mulțimii {a, b, c, d} sunt {a, b, c}, {a, b, d}, {a, c, d}, {b, c, d}.

Fie C_k mulțimea tuturor șirurilor care reprezintă *k-combinațiile* literelor din alfabetul englez. Fiecare șir din C_k este ordonat în ordine lexicografică crescătoare. De exemplu, C_{52} conține șirul:

'ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz' (întregul alfabet).

Fie *X* un șir nevid format din literele alfabetului englez. Considerând ordinea lexicografică a șirurilor, definim funcțiile *inferior* și *superior* după cum urmează:

- *inferior(k,X)* determină cel mai mare șir din C_k care nu este mai mare decât *X*; funcția *inferior(k,X)* nu este definită dacă *X* este mai mic decât cel mai mic șir din C_k.
- *superior(k,X)* determină cel mai mic șir din C_k care nu este mai mic decât *X*; funcția *superior(k,X)* nu este definită dacă *X* este mai mare decât cel mai mare șir din C_k.

De exemplu:
inferior(3,AB) este nedefinită,
superior(3,AB)=ABC;
inferior(3,a)=Zyz,
superior(3,a)=abc;
inferior(3,bde)=*superior*(3,bde)=bde;
inferior(3,xz)=xyz,
superior(3,xz) este nedefinită.

Scrieți un program care citește perechi de valori k X dintr-un fișier text și, pentru fiecare pereche, calculează *inferior*(k,X) și *superior*(k,X). Fiecare pereche k X se află pe o linie separată a fișierului text, k este un întreg, $1 \le k \le 52$, și X este un șir nevid format din cel mult 52 de litere. Fișierul de intrare conține date corecte.

Pentru fiecare pereche k X de valori, șirurile *inferior*(k,X) și *superior*(k,X) sunt afișate pe linii consecutive la ieșirea standard și sunt urmate de o linie vidă. Dacă șirurile *inferior*(k,X) și *superior*(k,X) nu sunt definite, vor fi afișate mesajele:

inferior = null, respectiv superior = null.

Exemplu:

Intrare: - **G.DAT**

```
3 AB
3 a
3 bde
3 bdeaa
3 xz
```

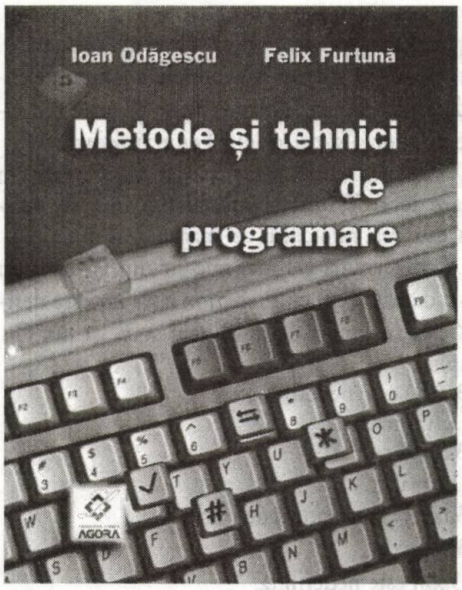
Ieșire: - ecran

```
inferior = null
superior = ABC

inferior = Zyz
superior = abc

inferior = bde
superior = bde

inferior = bde
superior = bdf

inferior = xyz
superior = null
```

Dacă *încă* nu

EȘTI PREGĂTIT PENTRU MÂINE,

trebuie să mergi la ȘCOALĂ, *trebuie* să ÎNVEȚI, îți *trebuie* ... un MANUAL.



Prof. univ. dr. ***IOAN ODĂGESCU***, este absolvent al *Facultăților de Cibernetică, Statistică și Informatică Economică* (***Academia de Studii Economice***) și *Matematică-Mecanică* (***Universitatea București***).

Cu o vechime de peste 25 de ani în învățământul superior de Informatică Economică, membru al Catedrei de Informatică Economică din ASE, a publicat multe lucrări de referință pe diferite domenii, cum ar fi:

În domeniul analizei numerice:
METODE NUMERICE ȘI SUBRUTINE
OPTIMIZAREA ALGORITMILOR

În domeniul informaticii:
VIRUȘI INFORMATICE ȘI BOMBE LOGICE – prima lucrare de acest gen din țară
LIMBAJUL COBOL FĂRĂ PROFESOR
SIMULAREA PROCESELOR ECONOMICE, GRAFICA INTERACTIVĂ PE CALCULATOARE PERSONALE
METODE ȘI TEHNICI DE PROGRAMARE – enunțuri, soluții, probleme propuse <<cartea galbenă>>)

Trebuie remarcat faptul că, domnul profesor a coordonat colectivul care a publicat prima lucrare pe plan național, întitulată OLIMPIADE NAȚIONALE ȘI INTERNAȚIONALE DE INFORMATICĂ

Domnul prof. Ioan Odăgescu este conducător de doctorat în specialitatea Informatică Economică. În cadrul acțiunilor științifice de promovare a tinerelor talente, de aproximativ 20 de ani se ocupă cu organizarea concursurilor de informatică și este autor a multor probleme date la concurs.

Este membru în consiliul ***Societății Române de Informatică*** (SOCROM INFO) și membru ***ACM*** (USA).

* * *

Asist. univ. drd. ***FELIX FURTUNĂ*** este absolvent al *Facultății de Cibernetică, Statistică și Informatică Economică* (***Academia de Studii Economice***, București), specializarea *Informatică Economică*. În prezent este cadru didactic la *Facultatea de Cibernetică Statistică și Informmatică Economică*, catedra de Informatică Economică.

Este coautor al lucrărilor:
METODE ȘI TEHNICI DE PROGRAMARE; OLIMPIADE NAȚIONALE ȘI INTERNAȚIONALE DE INFORMATICĂ
GRAFICA INTERACTIVĂ PE CALCULATOARE PERSONALE

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422] sau la Computer Press AGORA [Tel: 065-166516; 01-3309281].

BACALAUREAT 1999

Clara Ionescu

Începând cu acest număr GInfo va conține o secțiune nouă, denumită BAC, la cererea mai multor elevi și profesori. Vă prezentăm, pentru început programa de Bacalaureat pentru disciplina Informatică, aprobată de Ministerul Educației Naționale. Veți găsi în revistă și un posibil subiect de Bacalaureat, asemănător celui pe care elevii claselor a XII-a îl vor primi în luna iulie, în cadrul probei de informatică. Rezolvarea fiecărui subiect va fi publicată în numărul următor al revistei.

Programa de Bacalaureat pentru disciplina Informatică

Candidații trebuie să fie capabili să:

- recunoască o rezolvare corectă pentru o problemă dată
- analizeze o problemă și să aleagă metoda potrivită pentru rezolvarea ei
- reprezinte algoritmi de rezolvare a problemelor
- folosească corect elementele unui limbaj de programare
- transpună pașii unui algoritm în structuri de control specifice unui limbaj de programare
- organizeze informațiile în diferite structuri de date
- scrie programe sub formă modularizată
- utilizeze datele stocate pe suport extern
- folosească tehnici și metode de programare studiate în liceu
- enumere avantajele și dezavantajele metodelor de programare studiate în liceu
- aleagă metoda de programare potrivită pentru rezolvarea problemelor
- folosească structurile dinamice de date în rezolvarea problemelor

1. Algoritmi
   1.1. Noțiunea de algoritm, caracteristici
   1.2. Obiectele cu care lucrează algoritmii (date, variabile, expresii, operații)

2. Principiile programării structurate
   2.1. Structuri de bază
   - structurile liniară, alternativă și repetitivă
   - descrierea algoritmilor cu ajutorul schemelor logice și în pseudocod

3. Elementele de bază ale unui limbaj de programare (Pascal sau C)
   3.1. Vocabularul limbajului
   3.2. Constante
   3.3. Noțiunea de tip de dată
   3.4. Variabile; declararea variabilelor
   3.5. Definirea constantelor
   3.6. Expresii
   3.7. Citirea/scrierea datelor

4. Structuri de control
   4.1. Structura liniară; instrucțiunea de atribuire și instrucțiunea compusă
   4.2. Structura alternativă
   4.3. Structura repetitivă

5. Algoritmi simpli (care nu necesită utilizarea datelor structurate) Reprezentare în pseudocod și/sau scheme logice

5.1. Probleme care operează asupra cifrelor unui număr
5.2. Numere prime
5.3. Algoritmul lui Euclid
5.4. Șirul lui Fibonacci (fără **array**)

6. Tipuri structurate de date
6.1. Tipul tablou
6.2. Tipul înregistrare
6.3. Tipul mulțime

7. Aplicații cu tipuri de date structurate
7.1. Tablouri
7.1.1. Determinare minim/maxim
7.1.2. Ordonare (o metodă de ordonare: metoda bulelor, ordonare prin inserție, selecție, numărare)
7.1.3. Interclasare
7.1.4. Căutare binară
7.1.5. Prelucrări de șiruri de caractere
7.2. Înregistrări
7.3. Mulțimi
7.3.1. Operații cu mulțimi (reuniune, intersecție, diferență)

8. Subprograme
8.1. Domeniul de vizibilitate a identificatorilor
8.2. Dezvoltare ascendentă și dezvoltare descendentă a programelor
8.3. Proceduri
8.3.1. Declarare și apel
8.3.2. Parametri formali și parametri efectivi
8.3.3. Parametri transmiși prin valoare și parametri transmiși prin referință
8.4. Funcții; declarare și apel
8.5. Proceduri și funcții predefinite

9. Aplicații care necesită utilizarea subprogramelor
9.1. Scrierea modularizată a programelor
9.2. Programe ce evidențiază rolul parametrilor
9.3. Programe care utilizează funcții și proceduri predefinite

10. Fișiere. Aplicații
10.1. Noțiunea de fișier. Validare date
10.2. Fișiere text

11. Metoda Greedy
11.1. Prezentarea generală a metodei
11.2. Probleme

12. Metoda backtracking (iterativă sau recursivă)
12.1. Prezentare generală
12.1.1. Problema celor opt regine
12.1.2. Generarea partițiilor unui număr natural
12.1.3. Plata unei sume cu bancnote de valori date
12.1.4. Labirint
12.1.5. Algoritm de acoperire a tablei de șah prin săritura calului
12.1.6. Orice aplicație asemănătoare celor de mai sus

13. Recursivitate
13.1. Prezentare generală
13.2. Funcții recursive
13.2.1. Exemple la recursivitatea directă prin comparare cu metoda iterativă
- factorial
- șirul lui Fibonacci
- cel mai mare divizor comun
13.3. Proceduri recursive
- inversarea elementelor dintr-un șir
13.4. Metoda *Divide et impera*
- determinarea maximului și minimului unui șir
- căutare binară
- turnurile din Hanoi

14. Probleme de combinatorică
14.1. Generarea produsului cartezian
14.2. Generări de submulțimi
14.3. Generarea aranjamentelor
14.4. Generarea permutărilor

15. Structuri dinamice de date (alocare dinamică)
15.1. Tipul referință
15.2. Noțiunea de variabilă dinamică
15.3. Structuri de date înlănțuite
15.3.1. Liste
- operații elementare: inserare, căutare, eliminare element
- stive, cozi, liste dublu înlănțuite, liste circulare; operații specifice

Recomandări bibliografice

1. Ivașc, Cornelia și Prună, Mona - *Bazele informaticii clasa a X-a*, Editura Petrion, București, 1996
2. Ivașc, Cornelia și Prună, Mona - *Bazele informaticii, Caiet de laborator*, Editura Petrion, București, 1997
3. Negrescu, Liviu - *Limbajul C*, Editura Computer Libris Agora, Cluj, 1997
4. Niculescu, Stelian și Eftene, Sorin - *Algoritmi și limbaje de programare*, Editura Didactică și Pedagogică, București, 1995, 1997
5. Pătruț, Bogdan - *Învățați limbajul Pascal în 12 lecții*, Editura TEORA, București,1997
6. Rancea, Doina - *Limbajul Pascal*, Editura Libris, Cluj, 1993-1996
7. Tudor, Sorin - *Algoritmi și limbaje de programare*, Editura L&S, București, 1995
8. Tudor, Sorin - *Tehnici de programare*, Editura TEORA, București,1995

Un posibil subiect de Bacalaureat

Subiectul I:

a) Descrieți structura de date denumită *articol* sau *înregistrare.* (0,50 p)

b) Ce este pointerul? (0,50 p)

Subiectul II:

a) Precizați câți *octeți* va ocupa în memorie variabila ART declarată astfel:

Varianta în Pascal:

```
var ART:record Nume:string;
               Nr_tel:Longint;
               Varsta:Integer
    end;
```

Varianta în C++:

```
struct{
  char Nume[256];
  long Nr_tel;
  int Varsta} ART;
```

(0,25 p)

b) Precizați care este cel mai mare întreg cu semn care poate fi reprezentat pe *8 biți.* (0,25 p)

c) Precizați care este reprezentarea în *sistem hexazecimal* a numărului reprezentat în *sistem zecimal* prin 2457. (0,50 p)

d) Scrieți un algoritm care convertește în baza 10 un număr dat în baza 16. (1 p)

Subiectul III:

Care este efectul următorului algoritm?

```
Citește n
Pentru i=1,n execută Citește a_i
Sfârșit_pentru
Pentru i=1,[n/2] execută
  a_i ← a_i+a_n+1-i
  a_n+1-i ← a_i-a_n+1-i
  a_i ← a_i-a_n+1-i
Sfârșit_pentru
Pentru i=1,n execută Afișează a_i
Sfârșit_pentru
```

(1 p)

Subiectul IV:

Scrieți un program care implementează următorul algoritm de sortare:

- citește de la tastatură un șir de N numere întregi;
- împarte acest șir în două subșiruri, primul subșir conținând primele [N/2] elemente, iar al doilea restul elementelor;
- dacă unul dintre aceste subșiruri conține mai mult de un element, pentru ordonarea lui se folosește același algoritm, singura diferență fiind numărul de elemente;
- cele două subșiruri ordonate se interclasează și se obține șirul ordonat;
- afișează pe ecran șirul ordonat.

Nota redacției: După cum vedeți definiția problemei este una recursivă, deci este indicat să scrieți un program recursiv. (3 p)

Subiectul V:

Scrieți un program care citește un număr N din fișierul **INPUT.TXT** și N numere întregi din fișierul **NUMERE.TXT**, creează o *listă ordonată* care conține aceste numere în ordine crescătoare și scrie elementele listei în fișierul **OUTPUT.TXT**. (2 p)

Observații:

Toate subiectele sunt obligatorii.
Timp efectiv de lucru: 3 ore.
Se acordă un punct din oficiu.

Așteptăm rezolvările voastre!

Nu vă speriați, e mai ușor decât pare. *GInfo* este prietenul care vă va ajuta să ajungeți la "performanța" de a putea considera că acest subiect este foarte simplu. Nu trebuie decât să deveniți un cititor fidel al publicației noastre și examenul poate fi considerat "floare la ureche".

BAC '99

GAZETA DE INFORMATICĂ dorește să vină în sprijinul celor care doresc să se pregătească pentru acest examen, redacția oferindu-se să corecteze rezolvările voastre trimise pe adresa

Editura Computer LIBRIS Agora
Str. Universității 8/8
Cluj-Napoca

însoțite de talonul completat din ultima pagină a revistei.

Sperăm ca în acest mod să contribuim la pregătirea acelor elevi care vor susține proba de Informatică la examenul de Bacalaureat din acest an. Această serie de articole va sprijini și activitatea profesorilor care trebuie să-și pregătească elevii pentru examenul de Bacalaureat și nu numai. Toți cei care doresc să-și testeze cunoștințele în domeniul informaticii pot încerca să rezolve acest test.

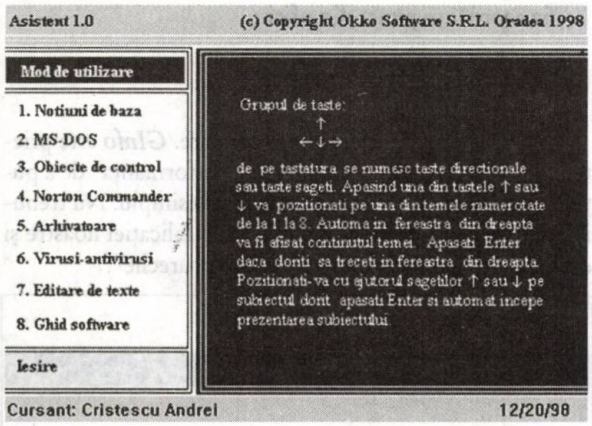
Succes!

SOFT educațional

Andraș Sorin

Care este cel mai la indemână instrument, posibil de folosit in scopul învățării "științei calculatoarelor"? Ați ghicit cu siguranță: este calculatorul și încă ceva: un soft adecvat!

Asistent 1.0



Asistent 1.0 se adresează celor care iau primul contact cu calculatorul, facilitând substanțial familiarizarea și însușirea noțiunilor elementare în informatică. Acest program se bazează pe experiența a patru ani în care am ținut cursuri de inițiere în informatică "copiilor" de la 8 până la 50 de ani, programul fiind conceput în așa fel încât utilizarea lui să nu fie mai complicată decât un joc obișnuit.

Imediat după lansarea în execuție a programului, utilizatorul este invitat să-și introducă numele.

Asistent 1.0 conține 99 de lecții organizate în opt capitole: Noțiuni de bază, Sistemul de operare MS-DOS, Obiecte de control, Norton Commander, Arhivatoare, Viruși-Antiviruși, Editare de texte, Ghid software. Pe parcursul capitolelor 1-7 se păstrează un mod unitar de prezentare: fiecare subiect este împărțit în pași distincți cu specificații ajutătoare (afișarea unei ferestre cu textul explicativ) la fiecare pas. După ce a fost realizată prezentarea unei lecții, utilizatorul are două opțiuni: "reia prezentarea lecției" sau "continuare". Dacă a fost aleasă opțiunea "continuare" utilizatorul este invitat să parcurgă singur pașii care au fost prezentați; orice greșeală atrage automat atenționarea utilizatorului că a greșit și urmează reluarea prezentării lecției. Dacă pașii au fost parcurși corect, utilizatorul este felicitat și poate trece la o lecție nouă. Capitolul "Ghid software" descrie pe scurt peste 140 de produse software grupate în 13 clase (Sisteme de operare, Inteligență artificială, Limbaje de programare, Sisteme de gestiune a bazelor de date, Calcul tabelar, Editoare de texte, Editoare grafice, Proiectare asistată, Rețele de calculatoare, Aplicații Internet, Utilitare, Programe educaționale, Jocuri) și prezintă un dicționar de extensii (peste 350) ordonate alfabetic.

La sfârșitul fiecărui capitol 1-7 există "Exerciții-test" care conțin 16-25 de exerciții prin care se verifică următoarele: recunoașterea și însușirea noțiunilor prezentate, capacitatea de recompunere a unor secvențe pentru a obține o definiție, o proprietate sau rezultatul unei probleme; posibilitatea realizării de asocieri; completarea unei definiții sau probleme din care lipsesc noțiunile cheie; capacitatea de comparare și decizie. După ce exercițiile au fost parcurse, se furnizează nota luată la fiecare exercițiu și media obținută. La ieșirea din program se afișează mediile obținute de utilizator la capitolele 1-7 (dacă au fost parcurse) din ziua respectivă, rezultate care pot fi listate și la imprimantă. După această afișare urmează prezentarea rezultatelor utilizatorului din prima zi când a folosit programul și până în ziua curentă, putându-se observa astfel performanțele realizate.

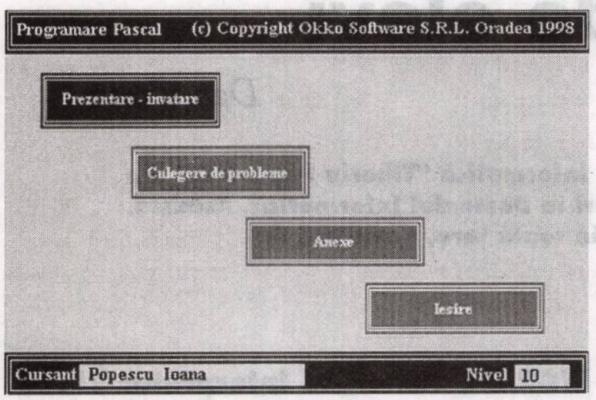
Asistent 1.0 poate rula pe un calculator minim 386 cu versiunea MS-DOS 5.0, monitor VGA color sau alb-negru. Programul funcționează și sub Windows 3.1, Windows 95 sau în rețea.

Programare Pascal

Programare Pascal este un soft educațional care se adresează celor care doresc să facă primii pași în programare. Programul are o interactivitate ridicată, este foarte sugestiv, cuprinzător, bogat în exemple și contraexemple, din aceste motive fiind util atât pentru însușirea cât și la recapitularea noțiunilor elementare (și nu numai) în programare.

din aceste motive fiind util atât pentru însușirea cât și la recapitularea noțiunilor elementare (și nu numai) în programare.

Programare Pascal este organizat în trei secțiuni: *Prezentare-învățare*, *Culegere de probleme*, *Anexe*. Cele trei secțiuni acoperă materia predată la clasa a IX-a, profil informatică.



Prezentare-învățare cuprinde 10 niveluri: *Algoritmi, Elemente de bază, Programare în TP, Structuri de control, Tipuri structurate, Proceduri și funcții, Tipul fișier, Unituri, Unit-ul Crt, Grafică și animație*. Executarea programului începe cu introducerea numelui și a unei parole, date de către utilizator. La primul contact cu programul utilizatorul este înregistrat implicit la nivelul 1. La fiecare nivel există "Exerciții-test" care conțin aproximativ 20 de exerciții de diferite tipuri. Dacă media obținută este mai mare decât 5 (sau o altă notă de trecere la nivelul următor, care poate fi fixată de către profesor) atunci utilizatorul are acces la nivelul 2, altfel este atenționat că nu stăpânește lecțiile primului nivel. Dacă utilizatorul a ajuns la nivelul 3, atunci la următoarea lansare în execuție (cu același nume și parola corespunzătoare) utilizatorul va continua de la nivelul 3, deci de la care a rămas, având acces la orice nivel inferior, dar fără drept de parcurgere a unui nivel superior. În momentul în care utilizatorul a ajuns la nivelul 10 va avea acces la orice nivel, chiar dacă ulterior nota luată la nivelul 1 este 4. Una din noutățile aduse în această secțiune ar fi, de exemplu, prezentarea în paralel (pe patru coloane) a următoarelor: *Enunțul problemei*, *Limbajul Natural*, *Schema Logică* și *Pseudocod*, ultimele două fiind generate pornind de la Limbajul Natural.

Culegere de probleme prezintă probleme rezolvate și probleme propuse (200/100) pentru: *Structuri de control, Șiruri și matrice, Șir de caractere, Tipul înregistrare, Tipul fișier, Proceduri și funcții, Unit*-uri, *Grafică*. Fiecare problemă rezolvată conține *Enunțul problemei, Limbajul Natural, Programul Pascal* și posibilitatea executării programului pentru problema respectivă. Problemele propuse conțin doar Enunțul problemei.

Anexe conține 10 anexe care oferă informații sau descrieri pentru: *Cuvinte rezervate, Identificatori predefiniți, Comenzi de editare, Taste uzuale, Diagrame de sintaxă, Comenzi de compilare, Mesaje de eroare* (compilare, rulare, grafică), *Codurile* ASCII, *Codurile tastelor, Sisteme de numerație*. Accesul, atât la *Anexe* cât și la *Culegere de probleme* este independent de nivelul la care se află utilizatorul, ele putând fi accesate fără restricție.

La ieșirea din program se afișează mediile obținute de utilizator la nivelurile 1-10 (dacă au fost parcurse) din ziua respectivă, rezultate care pot fi listate și la imprimantă. După această afișare urmează prezentarea rezultatelor utilizatorului din prima zi când a folosit programul și până în ziua curentă, putându-se observa performanțele realizate.

Programare Pascal poate rula pe un calculator minim 386 cu versiunea MS-DOS 5.0, monitor VGA color sau alb-negru. Programul funcționează și sub Windows 3.1, Windows 95 sau în rețea.

Domnul Andraș Sorin poate fi contactat prin e-mail la adresa:
okko@oradea.iiruc.ro

Informatica nu este o știință "l'art pour art"!

SOFT creat de elevi

Dan Crețu

În decembrie 1998 a avut loc la Liceul de Informatică "Tiberiu Popoviciu" din Cluj-Napoca a șasea Sesiune de Comunicări în Domeniul Informaticii. Această manifestare a reușit să reunească elevi din toată țara, pasionați de programarea aplicațiilor pe calculatoare.

Pentru a sublinia modul în care disciplina *informatică* își atinge obiectivele având caracter practic, Liceul de Informatică din Cluj organizează, de ani de zile, un simpozion în cadrul căruia sunt prezentate aplicații realizate de elevi. Participanții din acest an au prezentat aplicații bine realizate, cu o interfață corespunzătoare, demonstrând că atunci când au hotărât să urmeze cursurile unui liceu "cu profil", doreau să devină specialiști în acel domeniu, nu doar să-și "îmbogățească cultura generală".

Important mi se pare faptul că informatica în licee se concentrează pe algoritmică și prea puțin se insistă pe partea de operare, de programare a aplicațiilor, acest lucru defavorizând absolvenții acestor licee care, din diferite motive, nu s-au dus la facultate și sunt nevoiți să caute un loc de muncă. Dacă în timpul liceului nu au avut șansa să își lărgească domeniul de cunoștințe (limbaje de asamblare, programare sub Windows, medii vizuale de programare etc.) ei vor găsi extrem de greu un loc de muncă în care să își fructifice cunoștințele acumulate în liceu (Metode și tehnici de programare, Turbo Pascal sub Dos etc.). Acel atestat obținut după patru ani de "butonat" este mai puțin valoros decât cel pe care îl pot obține absolvenții unor forme de învățământ postliceal (2 ani) și chiar decât cel oferit de diversele firme care pretind că în șapte săptămâni cursanții lor au devenit "cunoscători" ai calculatorului.

Lăsând la o parte gândurile referitoare la conținutul învățământului de informatică, vă prezentăm produsul care a câștigat *premiul special*, oferit de sponsorul principal: firma *MicroInformatica*.

***Ovidiu Pitic*, autorul acestui produs, este elev în clasa a 12-a la Liceul de Informatică din Cluj.**

WORMie Command Interpretor V0.1B

WCI este un interpretor generic de comenzi, care permite integrarea unor comenzi de tip "linie de comandă" în programe utilizator.

Ceea ce merită a fi observat în legătură cu programul (`WCI.EXE`) este dimensiunea lui foarte mică, și anume 22880b în format executabil împachetat, și undeva pe la 50KB despachetat complet, acest lucru datorându-se în special scrierii lui în C++ și în limbaj de asamblare.

Comenzile `HELP` sau `?`

Odată cu lansarea programului, după afișarea titlului programului (titlul propriu-zis, versiunea lansată, precum și autorul și data realizării lui) va apărea un prompter. Din acest moment WCI este gata să primească orice comandă implementată anterior. Dacă utilizatorul dorește listarea comenzilor existente, o poate face cu una dintre comenzile `HELP` sau `?`, care vor avea ca efect afișarea în format *list* a comenzilor (în număr de 28 în această versiune). Lansarea uneia dintre aceste comenzi cu un parametru, va avea ca rezultat afișarea `HELP`-ului pentru comanda respectivă (dacă există), urmată de sintaxa caracteristică.

O altă comandă informativă, implementată este comanda `ABOUT` care oferă informații suplimentare despre program, cât și adresele de *S-Mail* și *E-Mail* ale autorului.

Comenzi legate de comenzile proprii ale sistemului de operare lansat (MS-DOS în acest caz):

Dintre acestea fac parte: `CD`, `COPY`, `CREATE`, `DIR`, `DISK`, `MKDIR`, `MOVE`, `RMDIR`, `RUN`, `WIN`; cele mai interesante vor fi prezentate succint în cele ce urmează.

Comanda `CD` este aproape identică cu cea oferită de Microsoft, dar aduce ca element nou posibilitatea schimbării într-un director specificat printr-un cuvânt rezervat (la fel ca toate comenzile ce folosesc direct structura de di-

rectoare). Cuvintele rezervate definite în această versiune sunt ROOT, FATHER, PARENT, GRANDFATHER. În cazul comenzii CD cu parametrul ROOT, se va schimba directorul curent astfel încât acesta să fie directorul rădăcină al discului curent, FATHER va urca cu un nivel în arborele de directoare, iar GRANDFATHER cu două niveluri. În cazul lansării comenzii fără parametri, se va afișa directorul curent.

Cuvintele rezervate precizate anterior pot fi modificate, pot fi adăugate altele noi prin modificarea unui tablou de **string**-uri în sursă.

Comanda DIR este o comandă, (după părerea autorului acestor note), mult mai flexibilă decât cea din MS-DOS pentru că într-o singură linie afișează toate informațiile posibile despre un fișier, implementarea fiind inspirată din structura comenzii ls din Linux. Este afișat și spațiul liber de pe discul curent.

Comanda DISK afișează, la lansarea fără parametri, numărul discurilor logice din sistem, precum și discurile disponibile utilizatorului. Lansând comanda cu un parametru - *literă*, se va schimba discul curent dacă acesta există și este accesibil, în caz contrar se afișează mesajul de eroare corespunzător.

Comanda RUN va lansa un program extern cu parametrii precizați după numele programului. La întoarcerea în "mediul" WCI, se va afișa codul de retur al programului lansat.

Comanda WIN va detecta lansarea interpretorului sub Windows(TM) sau DOS.

Comenzi care se ocupă de terminalul grafic: CLEAR, INK, PAPER, VIDEOFX

Comanda CLEAR șterge ecranul, lăsând doar prompterul în partea de sus a screen-ului.

Comanda INK urmată de un cuvânt rezervat - *culoare* (aceleași cu cele din C sau PASCAL: BLACK, BLUE, RED, GREEN, LIGHTGRAY etc., toate posibilitățile fiind implementate), realizează schimbarea culorii de scriere (textcolor) la culoarea specificată. Se fac verificări pentru culoarea de fond.

Comanda PAPER realizează schimbarea fondului de afișare în mod asemănător comenzii INK, dar pe lângă verificările legate de culoarea de scriere se verifică și posibilitatea utilizării culorii respective ca și culoare de fond.

Comanda VIDEOFX setează variabila internă VIDEOFX (ON sau OFF), după cum se precizează în parametrul comenzii. VIDEOFX ON are ca efect deplasarea continuă a trei benzi colorate pe verticala ecranului (afișarea lor se realizează în background, indiferent de ceea ce face utilizatorul, ele "dispărând" doar dacă se lansează un program extern). O lansare a comenzii fără parametri are ca efect afișarea stării variabilei.

Comenzi care se ocupă cu partea hard a sistemului: PROCESSOR, REBOOT, SLEEP, TYPEMATIC

Comanda PROCESSOR identifică tipul procesorului existent, în cazul procesoarelor care suportă comanda CPUID realizându-se o clasificare a procesorului: *(Non)Genuine*, *(Non)Intel*, *Family/Model/Stepping* și suportul *FPU*.

Comanda REBOOT realizează REBOOT-area sistemului astfel: fără parametri se realizează un "default REBOOT", în caz contrar se realizează o REBOOT-are la cald sau la rece (REBOOT WARM sau REBOOT COLD).

Comanda SLEEP, urmată de un număr de secunde în care procesorul să "doarmă", utilizează comanda internă HLT pentru a opri procesorul; cele mai noi modele (de la *Pentium(TM)* în sus) realizează oprirea doar a părților din procesor care nu lucrează, obținându-se după rapoartele celor de la Intel, o diminuare a temperaturii la suprafața procesorului cu până la 30°C.

Comanda TYPEMATIC setează rata de tastare a tastaturii, dacă aceasta este modificabilă via BIOS. Se poate seta întârzierea la tastare repetată la 250ms, 500ms, 750ms, respectiv 1s și rata de tastare de la 30cps la 2cps (cps = Characters Per Second).

Alte comenzi: COMP, PROMPT, SHOWPATH, EXIT și QUIT

Comanda COMP compară două fișiere, dar este disponibilă doar ca și comandă externă.

Comanda PROMPT modifică prompter-ul actual (cu un parametru-*șir de caractere* de lungime maximă 10) sau doar îl afișează (fără parametru).

SHOWPATH este, de asemenea, o comandă care setează o variabilă internă SHOWPATH. Dacă SHOWPATH este ON, prompter-ul va conține și calea curentă, pe lângă prompter-ul propriu-zis.

Comenzile EXIT și QUIT termină interpretorul de comenzi și ies în mediul inițial (DOS).

Utilizatorului i se oferă o rutină de citire a comenzii avansate, având posibilitatea continuării comenzii (cu TAB), reținerii ultimelor comenzi (HISTORY) și accesarea lor cu ajutorul tastelor *cursor up* și *cursor down*.

Mesajele date de interpretor sunt alese aleator, atenționându-se repetarea unei comenzi, încercarea lansării unei comenzi necunoscute, a unei comenzi prea lungi pentru a fi prelucrate (>100 caractere) sau a unui cuvânt prea lung (>32 caractere).

Acest produs este disponibil pentru download la adresa: www.ginfo.ro.

Autorul vă roagă să raportați orice "bug" sau, dacă aveți, orice sugestie/comentariu la adresa:

ubik@licj.soroscj.ro.

Executabilul, ca și sursele, pot fi cerute tot la adresa menționată.

Dan Crețu este student la Universitatea Tehnică din Cluj, Facultatea de Electrotehnică. Poate fi contactat prin e-mail la adresa: cretud@codec.ro

Numere prime și DIVIZIBILITATE

Mihai Oltean

Algoritmica este știința gândirii. De ce totuși trebuie să "convingem" cititorii de corectitudinea algoritmilor publicați, prezentând mereu și programele care codifică anumite raționamente? Să fie oare rezolvitorii de probleme prea comozi în sensul că le este "lene" să gândească? Vă punem la încercare... Iată câteva probleme rezolvate... doar algoritmic, respectiv matematic. Dacă sunteți curioși, implementați-le!

Problema 1
Săritura calului pe tabla de șah

Toată lumea cunoaște această problemă celebră a cărei soluție a fost expusă pentru prima dată de către *Euler*. Având în vedere titlul articolului, propun în cele ce urmează o generalizare a problemei. În primul rând trebuie specificat faptul că nu este vorba despre un cal obișnuit, ci despre unul care poate face sărituri puțin mai complicate, în sensul că numărul pătrățelelor peste care sare nu este 1, respectiv 2, ci p respectiv q. Problema care se pune este următoarea:

Determinați o condiție **necesară** *care trebuie să existe între* p *și* q*, astfel încât să existe un drum al calului care străbate toate pătrățele tablei, o singură dată.*

Rezolvare:

Observăm că în cazul clasic, (p,q∈{1,2}) avem întotdeauna soluție pentru o tablă de șah cu latura mai mare sau egală cu 5 pătrățele. Deducem de aici că s-ar putea ca p și q să fie prime sau să fie prime între ele. Ambele presupuneri sunt adevărate, dar cea de a doua este mai puternică decât prima, așa că o să o demonstrăm pe aceasta. Să presupunem că p și q nu sunt prime între ele și deci au un cel mai mare divizor comun, pe care-l vom nota cu d (d>1). Toate pătrățelele tablei care vor putea fi marcate de către săritura unui cal, vor avea coordonatele multiplu de d la care se adaugă coordonatele poziției de plecare.

Problema 2
Problema lui Joseph

Problema este la fel de cunoscută și are următorul enunț:

Se dau p *obiecte aflate pe un cerc și un număr natural nenul* q*. Marcarea obiectelor începe cu un obiect oarecare. Următorul obiect care se marchează este cel aflat la distanță de* q *obiecte față de ultimul marcat. Operația de* **marcare** *are sensul acelor de ceasornic și se continuă până în momentul în care un obiect este marcat de două ori. Se cere să se determine relația dintre* p *și* q*, astfel încât în final toate obiectele să fie marcate.*

Rezolvare:

Începem prin observația:

*Dacă se dau două numere naturale nenule (*p *și* q*), prime între ele, acestea se vor bucura de următoarea proprietate:*

$\{\hat{q}, \hat{2q}, \ldots, \hat{(p-1)q}\} = \{1, 2, \ldots, p-1\}$

unde prin $\hat{x}$ am notat restul împărțirii lui x la p.

Într-o primă etapă vom demonstra că numerele $\hat{q}$, $\hat{2q}, \ldots, \hat{(p-1)q}$ sunt distincte între ele. Presupunem, prin reducere la absurd, că ar exista două elemente ($\hat{iq}$ și $\hat{jq}$) egale. Atunci $\hat{(j-i)q}$ ar fi multiplu de p.

Astfel, am ajuns la o contradicție, deoarece j-i este mai mic decât p, iar p și q sunt prime între ele. Rezultă că toate cele p-1 numere sunt distincte, dar mai trebuie să demonstrăm că ele sunt și diferite de 0. Acest lucru se observă ușor, deoarece orice număr de forma i*q, (unde 1<=i<=p-1, iar p și q sunt prime între ele) nu se poate împărți exact la p.

Să demonstrăm acum și proprietatea inversă:

Dacă se dau două numere p *și* q, *astfel încât* { $\hat{q}$, $2\hat{q}$, ..., $(p-1)\hat{q}$ }={1,2,...,p-1}, *atunci* p *și* q *sunt prime între ele.*

Vom presupune prin reducere la absurd că p și q ar avea un cel mai mare divizor comun pe care îl vom nota cu d (d>1). Vom avea p=d*x și q=d*y. În cazul în care p<q rezultă că q*x este 0. În caz contrar (p>q) avem q*y=0, ceea ce contravine ipotezei. Astfel am demonstrat propoziția enunțată.

Într-o primă fază vom numerota cele p numere în ordinea acelor de ceasornic, cu numere de la 0 la p-1. La pasul întâi vom marca numărul 0. La pasul *w*, numărul marcat are valoarea egală cu restul împărțirii lui q*(w-1) la p. Dorim ca toate aceste resturi să fie diferite. Vom avea de executat exact p pași pentru a marca toate numerele. Conform observațiilor de mai sus, acest lucru este posibil doar dacă p și q sunt prime între ele.

În caz contrar se pot marca doar acele numere care sunt multipli ai cmmdc(p,q), la care se adaugă bineînțeles poziția de start.

O altă problemă echivalentă cu aceasta ar fi următoarea:

Problema 3
Problema paharelor de apă

Avem la dispoziție două pahare de apă A și B, cu capacitățile p *și* q, *(inițial goale). Operațiile posibile sunt următoarele:*

- *umplerea paharului A cu apă;*
- *golirea paharului A;*
- *umplerea paharului B cu apă;*
- *golirea paharului B;*
- *turnarea conținutului paharului A în paharul B;*
- *turnarea conținutului paharului B în paharul A.*

(Turnarea se face până când unul din pahare se golește sau celălalt se umple).

Problema cere să se determine ce relație trebuie să existe între p *și* q *astfel încât în urma unor operații de tipul celor descrise mai sus, în paharul de capacitate mai mare să se poată obține orice cantitate de apă.*

Rezolvare:

Să vedem de ce aceste probleme sunt echivalente. Marcarea elementului numerotat cu 0 de la ***problema lui Joseph*** este echivalentă cu faptul că paharul *A* (capacitatea p) este gol. Marcarea unui alt număr (după un salt de q unități) echivalează cu turnarea conținutului paharului *B* (de capacitate q) în paharul *A*. Dacă paharul *A* se umple, golirea lui și turnarea în el a apei care a rămas în *B* echivalează cu saltul peste obiectul cu numărul de ordine 0 de la problema lui Joseph. Astfel am arătat că cele două probleme sunt echivalente.

Condiția ca în paharul *A* să se poată obține orice cantitate de apă este ca numerele p și q să fie prime între ele. Dacă nu cerem ca în paharul *A* să poată fi obținută orice cantitate, ci o valoare specificată x, atunci este necesar ca x să fie multiplu de cmmdc(p,q). Deoarece inițial putem turna fie din paharul *A* în paharul *B*, fie invers, vom avea întotdeauna două soluții ale problemei: una mai lungă, necesitând mai multe operații și alta de lungime mai scurtă.

Completare:

Problema se poate generaliza la n pahare de apă, soluția rămânând aceeași.

Problema 4
Șirul Fibonacci și numerele prime

Fie șirul definit astfel:

F[0]=0, F[1]=1, F[n]=F[n-1]+F[n-2].

Există trei proprietăți extrem de interesante referitoare la acest șir:

1) oricare al k*-lea termen din șir se divide prin* F[k]*;*

2) dacă k *este număr compus, atunci și* F[k] *este tot un număr compus; (un număr este compus dacă nu este prim)*

3) dacă F[k] *este prim, atunci și* k *este prim, (excepție face* F[4]=3*);*

4) F[(m,n)]=(F[n],F[m]).

Rezolvare:

Se observă că numerele Fibonacci pare sunt F(3), F(6), F(9), F(12), ..., F(3k) sau mai precis, fiecare al treilea număr Fibonacci este multiplu de 2. Dar 2=F(3).

Se observă că numerele Fibonacci care sunt multiplu de 3 sunt: F(4), F(8), F(12), F(16), ..., F(4k) sau mai precis, fiecare al patrulea termen Fibonacci este multiplu de 3=F(4).

Regula generală ar fi prima dintre cele trei proprietăți, care exprimată "mai matematic" este:

F(nk) *este multiplu de* F(k) *pentru toate valorile lui* n *și* k=1,2,...

Această formă sugerează și metoda de demonstrare a corectitudinii afirmației, și anume *inducția matematică*, pe care pentru un k fixat o vom aplica după n. Pentru n=1 avem că F[k] divide pe F[k], deci propoziția este adevărată. Să presupunem că am demonstrat toate propozițiile pentru toate valorile mai mici și egale cu n-1. Vom demonstra afirmația pentru n. Pentru aceasta ne folosim de următoarea proprietate a numerelor Fibonacci, și anume:

F[m+n]=F[n-1]*F[m]+F[n]*F[m+1].

Deci F[n*k]=F[(n-1)*k+k]=F[(n-1)*k-1]*F[k]+F[(n-1)*k]*F[k+1].

În această relație F[k] și F[(n-1)*k] sunt multipli de F[k]. Rezultă că și F[n*k] este multiplu de F[k].

Proprietățile 2), 3) și 4) sunt consecințe directe ale proprietății 1), deci nu voi insista asupra lor. Proprietatea inversă proprietății 3) nu este valabilă, respectiv dacă k este prim, atunci și F[k] este prim. De exemplu:

F[19]=4181=113*37.

Problema 5
La cumpărături

La un magazin se găsesc n *tipuri de produse, fiecare având un cost dat* a[i], i=1..n. *În magazin intră o persoană care are asupra ei doar bancnote de* p *lei, unde* p *este un număr prim, și care dorește să cumpere cel puțin o bucată din fiecare produs. Datorită faptului că la caserie nu există bani pentru a se da rest, clientul va fi obligat să cumpere (dacă se poate) astfel încât să poată plăti exact, folosindu-se doar de bancnotele din buzunar (se presupune că are suficiente). Se cere să se determine, în cazul în care există soluție, câte bucăți cumpără din fiecare produs.*

Restricție:

Se pot cumpăra maxim p-1 *bucăți din fiecare produs, deoarece în caz contrar o soluție ar însemna să luăm exact* p *bucăți din fiecare.*

Rezolvare:

Ne vom baza pe următoarea observație:

Pentru x∈{1,...,p-1}, *vom avea*

{x̂, 2x̂, ..., (p−1)x̂}={1,2,...,p-1}.

Această remarcă reprezintă particularizarea observației făcute la problema 2, așa că nu am să reiau demonstrația.

Într-o primă etapă voi elimina din problemă acele produse pentru care costul este multiplu de p, deoarece dintre acestea, persoana respectivă va cumpăra doar câte o singură bucată. Mai rămân de cumpărat doar acele produse al căror cost nu este multiplu de p. Lucrurile se mai pot simplifica, dacă vom lucra cu resturile împărțirii numerelor a[i] la p. Fiecare rest va fi cuprins între 1 și p-1. Deoarece se pot cumpăra oricâte bucăți din fiecare, adică putem cumpăra o bucată, două bucăți, sau... (p-1) bucăți, pe baza observației de mai sus rezultă că putem obține orice rest cuprins între 1 și p-1. Dacă am avea un singur produs de cumpărat, problema nu ar avea soluție, deoarece indiferent câte bucăți am cumpăra din el, restul care ar trebui luat de la casă va fi cuprins între 1 și p-1.

Dacă, spre exemplu, am avea două produse de cumpărat, problema ar avea întotdeauna soluție, deoarece vom cumpăra din primul produs atâtea bucăți încât să avem restul 1, iar din celălalt produs vom cumpăra atâtea bucăți încât să avem restul p-1, deci suma lor va fi multiplu de p.

Dacă, spre exemplu avem de cumpărat p produse, atunci din fiecare vom lua atâtea bucăți, astfel încât restul împărțirii la p să fie 1, deci aceste resturi însumate vor fi exact p.

Concluzionând, problema are soluție în toate situațiile, mai puțin în cazul în care există exact un singur produs al cărui cost nu este multiplu de p. Problema se poate generaliza, în sensul că p nu trebuie să fie neapărat prim, (în schimb trebuie să fie neapărat prim cu fiecare dintre numerele a[i]). În acest caz soluția este aceeași și se bazează pe observația făcută la problema 2.

Aș mai dori să discut și o altă variantă a acestei probleme care, deși nu se încadrează în tematica propusă, o consider interesantă, și anume *eliminarea celor două restricții*: ca *din fiecare produs să se cumpere cel puțin o bucată* și ca p *să fie prim*. Rămâne și aici valabilă restricția ca din nici un produs să nu se poată cumpăra mai mult de p-1 bucăți. Mai mult, voi arăta că din nici un produs nu este nevoie să se cumpere mai mult de p **div** n bucăți (unde n este numărul de produse din magazin). Totodată problema are întotdeauna soluție dacă n>1. Dacă n=1 atunci problema are soluție doar dacă p și a[1] nu ar fi prime între ele. În acest caz (dacă notăm prin d cmmdc(p,a[1])) o soluție constă din a cumpăra p **div** d produse, pe care le-am putea plăti folosind a[1] **div** d bancnote. Dacă n>1 atunci soluția se aseamănă cu cazul în care se poate cumpăra maxim o bucată din fiecare produs și p=n. Reamintesc că în acea situație construiam sumele parțiale:

a[1],a[1]+a[2],a[1]+a[2]+a[3],...,
a[1]+a[2]+...+a[n].

Ajungeam la situația în care printre resturile împărțirii acestor sume la n existau două resturi egale sau exista unul egal cu 0. În situația noastră vom proceda asemănător, dar de data aceasta dispunem numerele pe cerc, iar sumele pe care le construim încep să zicem cu a[1] și se continuă în sensul acelor de ceasornic, până la suma de lungime p (care conține p termeni); evident, unii termeni vor fi luați de mai multe ori, dar nu mai mult de p **div** n+1 ori). Deci vom avea de a face cu p sume a căror p resturi rezultate în urma împărțirii la p sunt în intervalul 0..p-1, care este un interval de lungime p, deci fie că unul din resturi este 0, fie că două resturi sunt egale, din ambele situații rezultă imediat soluția.

Problema 6

Se dau 2p-1 *numere întregi (*p *este prim). Se cere să se determine o submulțime formată din* p *numere cu proprietatea că suma elementelor din această submulțime este divizibilă prin* p.

Rezolvare:

În primul rând trebuie să demonstrăm că dintr-o mulțime de p-1 numere naturale nedivizibile prin p se poate alege o submulțime a căror sumă să fie divizibilă prin x, oricare ar fi 1<=x<=p.

Algoritmul care determină această submulțime este dat și în [1], așa că doar am să îl schițez în continuare. Ne bazăm pe observația de la problema precedentă. Vom simplifica problema, în sensul că vom lucra doar cu resturile împărțirii numerelor date la p. Notăm aceste resturi cu y_i, i=1..p-1. Ne interesează să obținem fiecare rest între 1 și p-1 din sume (modulo p) ale numerelor din șirul y.

Pentru aceasta construim un nou vector boolean (atins), care pe poziția i, 1<=i<=p-1 reține dacă restul i a fost obținut din vreo sumă (modulo p) a numerelor date.

Inițial, toate elementele vectorului atins vor fi false. La fiecare pas vom marca un rest nou, și anume: la pasul i vom adăuga la sumele deja construite elementul y_i. Resturile (modulo p) ale sumelor deja construite sunt marcate în vectorul atins. Dacă vom adăuga elementul y_i la aceste resturi, vor rezulta noi resturi. Pe baza observației de la problema 5, avem garanția că toate resturile între 1 și p-1 vor fi atinse, deci există o submulțime de numere care, adunate vor da restul respectiv. Pentru a o reconstitui vom mai avea nevoie de încă un vector în care să fie reținută poziția de proveniență a fiecărui rest.

Să revenim la problema noastră. Fie:

a[1]<=a[2]<=....<=a[2p-1] resturile împărțirii celor 2p-1 numere la p. Considerăm numerele:

a[p+1]-a[2], a[p+2]-a[3], ..., a[2p-1]-a[p]. Dacă unul dintre aceste numere este egal cu zero, de exemplu a[p+j]-a[j], atunci a[j]=a[j+1]=...=a[p+j] constituie soluția problemei. Dacă nu există nici un astfel de număr egal cu 0, procedăm în felul următor:

Luăm suma elementelor a[1]+a[2]+...+a[p]. Dacă această sumă este egală cu 0 (evident modulo p) atunci ea este soluția problemei. În caz contrar, fie x restul împărțirii acestei sume la p. Aplicăm algoritmul de mai sus numerelor a[p+1]-a[2], a[p+2]-a[3], ..., a[2p-1]-a[p]. Am amintit că fiecare rest va putea fi obținut, deci și restul p-x. Adunăm numerele care însumate dau acest rest la a[1]+a[2]+...+a[p]. După efectuarea reducerilor evidente, vom avea soluția problemei.

Completare:

Se poate demonstra o proprietate și mai puternică, și anume că din 2p-1 numere se pot alege p, astfel încât suma lor să fie divizibilă cu p, fără restricția ca p să fie prim. De fapt se poate demonstra o proprietate chiar mai puternică, și anume: Din oricare a numere întregi se pot alege b numere a căror sumă se divide cu c, dacă și numai dacă b se divide cu c și a>=b+c-1. Pentru mai multe amănunte se poate consulta problema **M45** din [2].

Problema 7

Se dau două numere p *și* q. *Să se determine dacă există o mulțime de* p *numere prime mai mari decât 3, astfel încât suma pătratelor lor să fie egală cu* q *(modulo 12).*

Rezolvare:

De exemplu pentru p=13 și q=1 avem soluția: {5,7,13,17,19,29,31,37,41,43,47,53,59}. Suma pătratelor acestor numere este 16093=q (modulo 12).

Soluția acestei probleme este banală dacă se demonstrează o propoziție ajutătoare:

Pentru orice număr prim x, *mai mare decât 3 este adevărată proprietatea:*

x*x=1 (modulo 12).

Demonstrație:

Deoarece x este un număr prim, strict mai mare decât 3, atunci el poate avea una din formele:

12*k+1, 12*k+5, 12*k+7, 12*k+11.

Pentru x=12*k+1 avem (12*k+1)^2=1 (modulo 12)
Pentru x=12*k+5 avem (12*k+5)^2=1 (modulo 12)
Pentru x=12*k+7 avem (12*k+7)^2=1 (modulo 12)
Pentru x=12*k+11 avem (12*k+11)^2=1 (modulo 12)

Cu aceasta propoziția este demonstrată. Acum este evident că avem soluție doar în cazul în care p=q (modulo 12) și ea constă în alegerea la întâmplare a 12 numere prime.

Problema 8

Există și un algoritm care testează dacă o mulțime este inclusă într-o alta, folosind numere prime. Pentru aceasta să presupunem că cele două mulțimi (notate cu A și B) sunt din domeniul 1..100. Asociem fiecărui număr x (din domeniul stabilit anterior) al x-lea număr prim. Astfel, avem următoarele asocieri:

1 - 2
2 - 3
3 - 5
4 - 7
5 - 11
.....

Calculăm produsul numerelor prime asociate mulțimii A și produsul numerelor prime asociate mulțimii B. Dacă un rezultat (să zicem cel atașat mulțimii A) se divide cu rezultatul atașat celeilalte mulțimi (B) rezultă că B este inclusă în A.

Exemplu A={2,4,6,7}, B={4,7}. Mulțimea A are atașată mulțimea numerelor prime {3,7,13,17}, iar mulțimea B are atașată mulțimea {7,17}. Produsul elementelor mulțimii atașate lui A este 4641, iar cel al elementelor atașate mulțimii B este 119. Cel de al doilea produs divide pe primul, deci mulțimea B este inclusă în mulțimea A.

Inconvenientul acestui algoritm constă din faptul că trebuie efectuate înmulțiri și împărțiri cu numere mari.

Bibliografie:

1. *Proba de baraj pentru selecționarea lotului național*, problema "**La shop la Slatina**", *Gazeta de informatică*, Ediție specială, 1995
2. **Problema M45** din "*Probleme traduse din revista sovietică Kvant*", E.D.P., Buc, 1983.

Mihai Oltean este student la Facultatea de Matematică, secția Informatică, Universitatea Babeș-Bolyai, Cluj. Poate fi contactat prin e-mail la adresa: om7280@scs.ubbcluj.ro

Branch&Bound și BACKTRACKING

Ovidiu Gheorghioiu

Poate autorul acestui articol nu e singurul programator care a avut aversiuni față de metoda Branch&Bound. Ne-a mărturisit că dacă îi "pica" la vreun concurs o problemă de acest tip, de regulă prefera să încerce o altă metodă de rezolvare. A căutat o soluție alternativă la B&B și a găsit o metodă cunoscută: backtracking, sub o formă modificată. Dacă e implementată corect, metoda dă rezultate mult mai bune decât B&B chiar la problemele clasice, iar programele sunt mult mai scurte.

Ideea

O problemă tipică, la care de regulă se recomandă metoda *Branch&Bound* (de exemplu: Perspico), nu se poate rezolva cu metoda *backtracking* (în adâncime), deoarece nu se știe cât de departe trebuie coborât în arborele pozițiilor. *Propunem ca acest arbore să se genereze tot ca la metoda Branch&Bound, nivel cu nivel, la fiecare pas regenerând nivelurile anterioare.*

```
procedure Back(k:Integer);
begin
  if {solutie}
  then
  begin
    tipareste;
    Halt
  end;
  if k>max then Break;
        { pentru fiecare mutare valida:    }
        { - efectueaza mutarea;            }
        { - Back(k+1);                     }
        { - restaureaza starea anterioara; }
end;

Begin
  ...
  for max:=1 to max_posibil do
    Back(1);
  ...
End.
```

Complexitatea

Cred că deja se înțelege esența propunerii. Problema ***regenerării nivelurilor*** pare destul de complicată dar, un calcul simplu ne arată că nu afectează prea mult complexitatea, pentru că în general fiecare nivel este mult mai mare decât cele anterioare.

Dacă presupunem că, la fiecare pas, sunt în medie q posibilități, putem observa că cei doi algoritmi se comportă asemănător în ceea ce privește complexitatea. Ignorând deocamdată eventualele posibilități de optimizare, avem:

1) Nivelul de ordin n are dimensiunea q^n.
2) Metoda Branch&Bound generează nivelul n în:
   $O(q^1+q^2+\ldots+q^n)=O((q^{n+1}-1)/(q-1))=$
   $=O(q^{n+1}/(q-1))$.
3) Metoda backtracking generează la pasul n nivelul n în aceeași complexitate.

Dar în cazul aplicării metodei backtracking am generat și nivelurile anterioare, rezultă că de fapt avem o complexitate de ordinul:

$O((q^2+q^3+q^4+\ldots+q^{n+1})/(q-1))=O(q^{n+2}/(q-1))$,

adică timpul de executare în cazul folosirii metodei *backtracking este în medie de q ori mai mare decât în cazul folosirii metodei Branch&Bound.* Dar, ținând cont de faptul că valoarea lui q este constantă, rezultă că (din punct de vedere practic) ***au aceeași complexitate!***

Pe de altă parte, un pas în cazul metodei backtracking este mult mai simplu decât un pas în cazul metodei B&B, deoarece avem deja poziția în memorie și aceasta trebuie doar modificată, în schimb în cazul metodei B&B trebuie

împachetată/despachetată, trebuie refăcute diferite calcule etc. Dacă ținem cont de marele dezavantaj al metodei Branch&Bound - și anume faptul că este mare consumatoare de ***memorie***, rezultă că observația că *soluția de mai sus funcționează mult mai bine decât Branch&Bound*, nu trebuie să ne mire.

Totuși, metoda Branch&Bound nu trebuie "aruncată la coș", deoarece are avantajele sale cunoscute: de exemplu, dacă spațiul pozițiilor este limitat, aplicând metoda B&B vom obține soluții mai rapide. În plus, se pot tria soluțiile în memorie după o funcție de cost (în general euristică). Dar această modalitate funcționează numai în cazul în care funcția este "foarte bună" și nici chiar atunci nu este fiabilă pentru valori prea mari, deoarece în procesul trierii se pierd soluții posibile, datorită limitelor memoriei.

Evident, și metoda propusă mai sus poate fi *optimizată*, folosind "artificii de programare" cunoscute. Avantajul constă în faptul că la orice pas se știe cât de departe se poate avansa în spațiul soluțiilor. Dacă este posibilă estimarea corectă a ***numărului aproximativ de mutări*** se poate ieși dintr-o ramură care probabil nu conduce la soluție. Această optimizare a fost experimentată pe problema Perspico și rezultatele confirmă "prezicerile": un program neoptimizat găsește soluția în cazul în care aceasta este o secvență de cel mult 20-22 de mutări, iar unul optimizat găsește soluția chiar dacă aceasta este o secvență de până la 50 de mutări!

În continuare vă prezentăm sursele (problema Perspico rezolvată cu Branch&Bound apoi, rezolvată cu metoda propusă):

Listing BB.PAS

```
{$g+,s-,r+}
program Perspico_Branch_n_Bound;
type position=array[1..4,1..4] of Byte;
     leg=^nod;
     nod=record
           nm,sum,x,y:Byte;
           pm:Char;
           pos:position;
           next:leg
         end;
var a:position;
    l:array[0..15] of Byte absolute a;
    t:array[0..200] of leg;
    i,j,k,x,y,nm,sum,min,max:Integer;
    pm:Char;
    p,q:leg;
    f:Text;
    meminit:Longint;

procedure citeste;
begin
  Assign(f,'PERSPICO.IN');
  Reset(f);
  for i:=1 to 4 do
    for j:=1 to 4 do
    begin
      Read(f,a[i,j]);
      if a[i,j]=0
      then
      begin
        x:=i;
        y:=j
      end
      else
        Inc(sum,Abs(i-(a[i,j]+3) shr 2)+
                  Abs(j-(a[i,j]-1) and 3-1))
    end;
  Close(f)
end;

function ex_sol:Boolean;
var s:Integer;
begin
  s:=x+y;
  for i:=0 to 14 do
    for j:=i+1 to 15 do
      if l[j]>l[i] then Inc(s);
  ex_sol:=Odd(s)
end;

procedure scrie(da:Boolean);
begin
  Assign(f,'PERSPICO.OUT'); Rewrite(f);
  if da
  then
  begin
    Writeln(min,' mutari');
    Writeln(f,min);
  end
  else Writeln('Nu exista solutie!');
  Close(f);
  Halt
end;

procedure store_pos(name:Char);
begin
  if (maxavail>=sizeof(nod)+50) and
     (meminit-maxavail<16000000)
  then New(q)
  else
    if nm+sum<max
    then
    begin
      q:=t[max];
      t[max]:=t[max]^.next;
      while (t[max]=nil) and (max>nm+sum) do
        Dec(max)
    end
    else Exit;
```

```
  q^.next:=t[nm+sum];
  t[nm+sum]:=q;
  if max<nm+sum then max:=nm+sum;
  if sum=0 then scrie(true);
  q^.nm:=nm;                          { Pack }
  q^.sum:=sum;
  q^.x:=x;
  q^.y:=y;
  q^.pm:=name;
  q^.pos:=a
end;

procedure try_move(dx,dy:Integer;
name:Char);
var tx,ty:Integer;
begin
  a[x,y]:=a[x+dx,y+dy];
  a[x+dx,y+dy]:=0;
  tx:=(a[x,y]+3) shr 2;
  ty:=(a[x,y]-1) and 3+1;
  Inc(sum,Abs(x-tx)-Abs(x+dx-tx));
  Inc(sum,Abs(y-ty)-Abs(y+dy-ty));
  Inc(x,dx);
  Inc(y,dy);
  Inc(nm);
  store_pos(name);
  Dec(nm);
  Dec(x,dx);
  Dec(y,dy);
  Dec(sum,Abs(x-tx)-Abs(x+dx-tx));
  Dec(sum,Abs(y-ty)-Abs(y+dy-ty));
  a[x+dx,y+dy]:=a[x,y];
  a[x,y]:=0
end;

procedure rezolva;
begin
  if not ex_sol then scrie(false);
  store_pos('A');                     { Init }
  min:=sum; max:=sum;
  while true do                       { Run  }
  begin       { se va iesi cand se executa }
              { Halt in procedura scrie    }
    if Random(1000)=0
    then
      Write(#13,min,' ',max,' ',maxavail,' ');
    while t[min]=nil do Inc(min);
    p:=t[min];
    t[min]:=p^.next;                  { Delete }
    nm:=p^.nm;                        { Unpack }
    sum:=p^.sum;
    x:=p^.x;
    y:=p^.y;
    pm:=p^.pm;
    a:=p^.pos;
    if (x>1) and (pm<>'U')            { Expand }
    then try_move(-1,0,'D');
    if (x<4) and (pm<>'D')
    then try_move(1,0,'U');
    if (y>1) and (pm<>'L')
    then try_move(0,-1,'R');
    if (y<4) and (pm<>'R')
    then try_move(0,1,'L');
    Dispose(p)
  end
end;

Begin
  meminit:=maxavail;
  Writeln(meminit);
  citeste;
  rezolva
End.
```

Vă prezentăm în continuare textul sursă al programului care implementează algoritmul propus, cu optimizarea posibilă, menționată.

Listing PERSPICO.PAS

```
{$s-,g+}
Program Perspico_optimizat;
var a:array[1..4,1..4] of Integer;
    l:array[0..15] of Integer absolute a;
    x,y,max,i,j,sum,smin:Integer;
    s:string;
    f:Text;

procedure citeste;
begin
  Assign(f,'PERSPICO.IN'); Reset(f);
  for i:=1 to 4 do
    for j:=1 to 4 do
    begin
      Read(f,a[i,j]);
      if a[i,j]=0
      then
      begin x:=i; y:=j end
      else
        Inc(sum,Abs(i-(a[i,j]+3) shr 2)+
            Abs(j-(a[i,j]-1) and 3-1));
    end;
    Close(f)
end;

function ex_sol:Boolean;
var s:Integer;
begin
  s:=x+y;
  for i:=0 to 14 do
    for j:=i+1 to 15 do
      if l[j]>l[i] then Inc(s);
  ex_sol:=Odd(s)
```

```
end;
procedure scrie(da:Boolean);
begin
  Assign(f,'PERSPICO.OUT'); Rewrite(f);
  if da
  then
  begin
    Writeln('Ok!');
    Writeln(f,max);
    s[0]:=Chr(max);
    Writeln(f,s)
  end
  else Writeln('Nu exista solutie!');
  Close(f);
  Halt
end;

procedure back(k:Integer;p:Char);
  procedure
try_move(dx,dy:Integer;name:Char);
  var tx,ty:Integer;
  begin
    a[x,y]:=a[x+dx,y+dy];
    a[x+dx,y+dy]:=0;
    tx:=(a[x,y]+3) shr 2;
    ty:=(a[x,y]-1) and 3+1;
    Inc(sum,Abs(x-tx)-Abs(x+dx-tx));
    Inc(sum,Abs(y-ty)-Abs(y+dy-ty));
    Inc(x,dx);
    Inc(y,dy);
    s[k+1]:=name;
    back(k+1,name);
    Dec(x,dx);
    Dec(y,dy);
    Dec(sum,Abs(x-tx)-Abs(x+dx-tx));
    Dec(sum,Abs(y-ty)-Abs(y+dy-ty));
    a[x+dx,y+dy]:=a[x,y]; a[x,y]:=0
  end;
begin
  if k+sum>max then Exit;
  if k=max then scrie(true);
  if (x>1) and (p<>'U')
  then try_move(-1,0,'D');
  if (x<4) and (p<>'D')
  then try_move(1,0,'U');
  if (y>1) and (p<>'L')
  then try_move(0,-1,'R');
  if (y<4) and (p<>'R')
  then try_move(0,1,'L')
end;
procedure rezolva;
begin
  max:=sum;
  if ex_sol
  then
    while true do
    begin
      Write(#13'Caut solutie in ');
      Write(max,' mutari... ');
      back(0,'A'); Inc(max,2)
    end
  else scrie(false)
end;

Begin
  citeste; rezolva
End.
```

Vă propunem să vă "jucați" cu cele două alternative de rezolvare, experimentându-le, comparându-le pentru datele din următorul fișier de intrare:

PERSPICO.IN

```
6 2 8 1
13 5 7 4
14 15 3 11
9 0 10 12

2 0 7 1
6 8 13 4
14 11 12 10
3 5 9 15

2 6 8 4
5 3 1 15
14 13 11 12
7 10 0 9

8 2 5 11
1 6 9 10
3 15 0 7
12 14 4 13

5 3 13 15
6 14 10  8
7 12 11  2
4  9  1 0

1 2 3 4
5 6 0 7
9 14 15 8
13 12 10 11

8 2 5 11
1 6 9 10
3 15 0 7
12 14 4 13
```

Ovidiu Gheorghioiu este student la Universitatea Politehnică din București. Poate fi contactat prin e-mail la adresa: ovy@iname.com.

Din secretele "creierului" calculatorului (2)

MICROPROCESOARE

Mihai Scorțaru

Continuarea promisă...
Programarea în cod mașină sau în limbaj de asamblare poate oferi foarte multe satisfacții. De exemplu, vă puteți "răzbuna" pe îngrădirile impuse de *programarea structurată* sau *obiectuală* și puteți exploata o serie de facilități, accesibile doar la acest nivel.

Instrucțiuni ale microprocesorului

Setul de instrucțiuni ale microprocesorului reprezintă totalitatea instrucțiunilor pe care un microprocesor le recunoaște și pe care acesta le poate executa. Calitatea setului de instrucțiuni este cea care determină, în ultimă instanță, performanțele microprocesorului. Aceste performanțe mai depind - într-o mai mică măsură - și de numărul regiștrilor interni, de frecvența maximă de tact, de spațiul de adresare, de lățimea magistralei de adrese etc. Cu cât setul de instrucțiuni este mai variat și cu cât tehnicile de adresare sunt mai numeroase, cu atât calculatorul va putea fi folosit pentru rezolvarea în mod flexibil și eficient a mai multor probleme, iar microprocesorul în cauză va fi mai performant.

Reprezentarea instrucțiunilor în cod mașină

O instrucțiune trebuie să conțină *informații* asupra operațiilor care trebuie efectuate, precum și *datele* sau *adresele datelor implicate în aceste operații*.

În cazul microprocesoarelor convenționale (8086/8088) o instrucțiune ocupă între 1-6 bytes (majoritatea ocupă 2, 3 sau 4 bytes).

În cazul microprocesoarelor Intel o instrucțiune este compusă din:

- o zonă de un byte care conține ***codul*** operației care trebuie efectuată (*CODOP*);
- o zonă de un byte care conține informații privind ***modul de adresare*** (*MOD R/M*);
- o zonă de cel mult patru octeți care conține ***datele*** implicate în instrucțiune sau ***adresa*** acestor date (*OFFSET*).

Zona *OFFSET* poate lipsi în cazul unor instrucțiuni care nu ocupă decât 2 bytes (corespunzători zonelor *CODOP* și *MOD R/M*). În cazul altor instrucțiuni lipsește și zona *MOD R/M* (ele nu ocupă decât un singur byte). Zona *CODOP* nu poate lipsi din nici o instrucțiune, ea fiind cea care indică ce operații trebuie efectuate. În cazul microprocesoarelor 80286 și a celor ulterioare, unele instrucțiuni pot avea o zonă *CODOP* de doi bytes.

Instrucțiunile sunt întotdeauna încărcate din segmentul de cod curent, registrul `IP` conținând deplasamentul instrucțiunii următoare față de baza segmentului. Majoritatea datelor sunt păstrate în segmentul de date curent, dar există posibilitatea căutării unei date în orice alt segment adresabil. Deplasamentul unei variabile din memorie este calculat în funcție de modul de adresare specificat în zona *MOD R/M* a instrucțiunii, rezultatul purtând denumirea de ***adresă efectivă (AE)***.

Spre deosebire de restul variabilelor, ***șirurile*** se bucură de un "tratament" special, ele fiind adresate într-un mod diferit (față de celelalte variabile). Deplasamentul ***operandului sursă*** al unei instrucțiuni care lucrează cu șiruri se află în registrul `SI`, iar deplasamentul ***operandului destinație*** este luat din registrul `DI`. Operandul sursă se află, de obicei, în segmentul de date curent, dar se poate găsi în orice alt segment; în schimb, operandul destinație se află întotdeauna în extrasegmentul curent. Regiștrii `SI` și `DI` sunt ajustați automat după fiecare instrucțiune de acest tip.

În cazul în care se dorește adresarea unei locații din unul din segmentele curent adresabile, instrucțiunea trebuie să fie ***prefixată*** de un octet care specifică registrul segment al locației respective.

Faptul că memoria este structurată pe segmente face posibilă scrierea de programe independente de poziția lor în memorie (sunt ***relocabile dinamic***). Pentru ca un program să fie ***relocabil dinamic*** el nu are voie să modifice regiștrii săi de segment și nici să facă transferuri directe la o locație care nu se află în segmentul său de cod. Aceasta înseamnă că toate deplasamentele din program trebuie să fie relative la niște valori fixe conținute în regiștrii de segment. Dacă aceste condiții sunt îndeplinite, programul poate fi mutat în orice zonă a memoriei disponibile, atâta timp cât regiștrii săi de segment sunt actualizați de către sistemul de operare, în sensul că referințele acestora sunt modificate astfel încât să indice noua adresă de bază.

Moduri de adresare

"Unitatea elementară" care este executată de către microprocesor se numește ***instrucțiune***, deci putem spune că microprocesorul lucrează cu instrucțiuni. La rândul lor instrucțiunile operează cu date numite *operanzi*. Metoda folosită pentru a comunica microprocesorului adresa efectivă la care este memorată o anumită informație se numește ***mod de adresare***.

Majoritatea instrucțiunilor lucrează cu cel puțin un operand, aceștia putându-se afla chiar în corpul instrucțiunii, în regiștri, în memoria internă sau în porturile de intrare/ieșire. Modul de adresare al unui anumit operand depinde de locul în care este el memorat.

Am amintit anterior că o instrucțiune poate conține *o anumită dată care va fi folosită de microprocesor în execuția instrucțiunii* sau poate conține numai *adresa de memorie a datei respective*. Dacă o instrucțiune înglobează în corpul său un operand, modul de adresare folosit poartă denumirea de ***adresare imediată***.

Atenție!!! Nu confundați ***corpul și codul unei instrucțiuni.***

Operanzii de acest tip se numesc ***operanzi imediați*** sau ***date imediate*** și sunt date constante care pot fi reprezentate pe 8 sau 16 biți (se poate ajunge până la 32 de biți pentru microprocesoarele apărute în ultima vreme) și care sunt conținute în instrucțiune. Acești operanzi pot fi adresați foarte rapid, deoarece ei se află în aceeași zonă de memorie în care se află și instrucțiunea și, din această cauză, nu mai este nevoie de efectuarea unor operații suplimentare pentru obținerea lor. Din cauza limitării numărului de biți care pot fi folosiți pentru reprezentarea unui *operand imediat*, este evident că nu pot fi *date imediate* constantele *șiruri de caractere* care conțin mai mult de 2 caractere (4 sau 8 la microprocesoarele noi).

Există anumite limitări ale acestui tip de operanzi, cea mai importantă fiind faptul că acești operanzi nu pot fi decât valori constante și pot servi numai ca operanzi sursă. Operandul destinație nu poate fi o dată imediată deoarece destinația trebuie să reprezinte un loc de depozitare a unei noi informații și nu o constantă.

În cazul în care se adresează o dată păstrată într-un registru, modul de adresare se numește ***adresare la regiștri;*** operanzii adresați în acest mod se numesc ***operanzi-registru***. Acest mod de adresare constă în folosirea valorii efective care se află în interiorul registrului. Instrucțiunile care folosesc numai operanzi-registru sunt compacte și mult mai rapide, deoarece toate operațiile sunt executate în interiorul unității centrale, nemaifiind necesare transferuri suplimentare de date pe magistrală. Pot fi regiștri atât operanzii sursă cât și operanzii destinație. Regiștrii care pot fi folosiți ca operanzi într-o instrucțiune sunt **AX, BX, CX, DX, AH, BH, CH, DH, AL, BL, CL, DL, SP, BP, SI, DI, CS, SS, DS** și **ES**. Pentru microprocesoarele 80386 și pentru cele apărute ulterior mai pot fi folosiți regiștrii **EAX, EBX, ECX, EDX, ESP, ESI, EDI, FS** și **GS**.

Multe instrucțiuni pot opera asupra datelor din memorie; microprocesorul are sarcina de a calcula adresele acestor date. Adresele calculate în acest mod poartă denumirea de ***adrese efective (AE)*** și determinarea lor depinde de modalitatea în care este specificat operandul. Trebuie precizat faptul că operațiile *memorie-la-memorie* nu sunt admise; pentru a transfera o anumită dată dintr-o zonă de memorie în alta este nevoie de un registru intermediar. În cazul **adresării directe**, *AE* se obține adăugând la valoarea de bază a segmentului de date un deplasament de 16 biți. Valoarea de bază a segmentului de date este păstrată în registrul **DS**, în timp ce în corpul instrucțiunii apare numai valoarea deplasamentului. Adresarea unui operand de acest tip poate fi realizată și cu ajutorul regiștrilor, acest mod de adresare purtând denumirea de ***adresare indirectă prin regiștri***. În acest caz, adresa efectivă este calculată pe baza informației preluate dintr-unul din regiștrii de bază sau de index: **BX, BP, SI** sau **DI**.

Tipuri de ***adresare indirectă***:

1. *Adresarea bazată*
   Conținutul lui *AE* reprezintă suma dintre conținutul registrului **BX** sau **BP** și deplasamentul de după codul instrucțiunii; registrul utilizat în acest scop se numește ***registru de bază***. Dacă acesta este **BX** atunci registrul de segment se consideră a fi **DS**; în caz contrar registrul de segment este considerat registrul **SS**. Dacă nu este specificat nici un deplasament, *AE* se calculează folosind doar conținutul registrului de bază și modul de adresare poartă denumirea de ***adresare bazată implicită***.
2. *Adresarea indexată*
   AE conține suma dintre conținutul registrului **DI** sau **SI** și deplasamentul ce urmează după codul instrucțiunii; registrul folosit în acest scop poartă denumirea de ***registru de index***. Dacă nu este specificat nici un deplasament, *AE* se calculează folosind doar conținutul registrului de index, iar modul de adresare poartă denumirea de ***adresare indexată implicită***. .
3. *Adresarea bazată și indexată*
   AE conține suma dintre conținutul unui registru de bază, a unui registru de index și un deplasament. Registrul de bază "dictează" și registrul de segment implicit (dacă registrul de bază este **BX**, registrul de segment este **DS**, în caz contrar registrul de segment este **SS**).

Pentru a adresa porturile de intrare/ieșire sunt utilizate două moduri de adresare:

1. *Adresare directă* - portul este un *operand imediat* pe 8 biți, ceea ce permite adresarea porturilor numerotate de la 0 până la 255.
2. *Adresare indirectă* - este similară cu adresarea indirectă a memoriei prin regiștri. Numărul portului este preluat din registrul **DX** și pot fi adresate porturi numerotate de la 0 la 65535.

De fiecare dată când o instrucțiune este încărcată în memorie, adresa locației de memorie în care se află memorată instrucțiunea este calculată ca o sumă dintre adresa segmentului de cod (păstrată în registrul `CS`) și un deplasament (păstrat în registrul `IP`). În mod normal conținutul registrului `IP` este incrementat după executarea fiecărei instrucțiuni. Există instrucțiuni care permit modificarea conținutului acestui registru prin următoarele moduri de adresare:

- ***adresare relativă (salt intrasegment):*** la conținutul registrului `IP` se adună un deplasament reprezentat pe 8 sau 16 biți ca un număr cu semn (valoarea păstrată în acest registru poate să scadă sau să crească). Această operație nu alterează conținutul registrului segmentului de cod `CS`.
- ***adresare directă (salt intersegment):*** în regiștrii `CS` și `IP` sunt încărcate date ce reprezintă noua adresă. Saltul se face între segmente de cod diferite (conținutul registrului `CS` este alterat).
- ***adresare indirectă:*** adresa instrucțiunii următoare se obține dintr-o dată citită din memorie, pentru a cărei citire pot fi folosite oricare din modurile de adresare prezentate anterior. Pentru modul indirect de adresare există două opțiuni: pot fi citiți 16 biți (se modifică doar conținutul registrului `IP`) sau pot fi citiți 32 de biți (va fi alterat și conținutul registrului `CS`).

Instrucțiuni ale limbajului de asamblare

Nu ne propunem să prezentăm în detaliu un limbaj de asamblare dar, pentru a ilustra mai clar modul în care funcționează microprocesorul, vă vom prezenta cele mai simple modalități de a da anumite comenzi acestuia.

Alfabetul unui limbaj de asamblare conține:
- literele alfabetului latin (`'A'..'Z'; 'a'..'z'`)
- cele 10 cifre (`'0'..'9'`)
- semnele operațiilor aritmetice (`'+', '-', '*'` și `'/'`)
- delimitatori (spațiu, tab, <CR>, ';')
- semne de punctuație (`'.', ',', '(', ')', '[', ']'`)
- alte caractere speciale (`'_', '@', '$', '?'`).

Cuvintele limbajului sunt formate din cel mult 31 de litere, cifre și caractere speciale. Primul caracter este întotdeauna o literă sau un caracter special. O categorie de cuvinte o reprezintă ***cuvintele rezervate*** (***cuvinte mnemonice***) ce cuprind numele instrucțiunilor, pseudoinstrucțiunilor, operatorilor, regiștrilor, precum și cuvintele "$" și "@". În cazul ***cuvintelor utilizator,*** primele 6 caractere trebuie să fie distincte, deoarece în momentul link-editării acestea sunt de obicei trunchiate la șase caractere.

Propozițiile limbajului reprezintă succesiuni de cuvinte și pot fi de trei tipuri:

1. ***Instrucțiunile*** - formează categoria cea mai importantă care cuprinde propozițiile ce descriu acțiuni pe care trebuie să le execute procesorul. Între acestea și instrucțiunile cod-mașină există o corespondență de ***unu la unu***;
2. Propoziții care ***descriu date***, respectiv zone de memorie de date;
3. ***Pseudoinstrucțiunile*** (***directive***) - nu se adresează microprocesorului ci numai asamblorului, indicând modul în care trebuie tratate propozițiile care urmează.

Propoziția poate fi privită ca o combinație între operații, operanzi și comentarii care definesc codul obiect care va fi creat în momentul asamblării. O linie a codului sursă poate conține o singură propoziție și nu poate depăși 1024 de caractere.

Sintaxa generală a unei propoziții este:
[etichetă] [operație] [operanzi] [; comentariu]
unde:
- ***eticheta*** este un identificator care etichetează propoziția pentru ca aceasta să poată fi accesată prin nume de către alte propoziții;
- ***operația*** este un cuvânt rezervat care definește acțiunea propoziției;
- ***operanzii*** definesc datele ce vor fi operate de propoziție;
- ***comentariul*** este un text care nu afectează acțiunea propoziției, fiind ignorat de către asamblor.

Instrucțiunile au rolul de a dirija microprocesorul. Ele sunt transformate în cod obiect în momentul asamblării, acest cod obiect controlând comportamentul microprocesorului.

Pseudoinstrucțiunile au rolul de a dirija asamblorul. Ele specifică modul în care asamblorul va genera codul obiect în momentul asamblării și nu se regăsesc în codul obiect rezultat în urma asamblării.

Operanzii pot fi compuși din simboluri, constante, expresii și regiștri.

Comentariile reprezintă descrieri ale codului sursă și sunt folosite exclusiv pentru documentarea programului sursă. Textul de pe o linie care urmează după caracterul *';'* este considerat comentariu. Pentru a insera comentarii care ocupă mai multe linii trebuie folosit caracterul *';'* pe fiecare linie sau, pe linie separată directiva `comment`. Sintaxa acestei directive este:

`comment` *delimitator [text]*
text
delimitator [text]

Asamblorul ignoră textul care apare între primul delimitator și linia care conține al doilea delimitator. Iată și un exemplu:

```
comment + Semnul plus este un delimitator
  Acest text este ignorat.
+ Si acest text este ignorat.
```

Structura unui program simplu

Vă prezentăm în continuare tradiționalul program care afișează mesajul: *Hello World!*. Pentru a scrie acest program utilizați orice editor de texte care permite salvarea textului în format ASCII. Vom presupune că ați denumit fișierul creat `HELLO.ASM`.

```
stiva segment stack 'STACK'
  DB 100h dup(?)
stiva ends
date segment
mesaj DB 'Hello World!', 13, 10, '$'
date ends
      assume ss:stiva, ds:date, cs:codp
codp segment
start: mov ax, date
       mov ds, ax
       mov ah, 9
       mov, dx, OFFSET mesaj
       int 21h
       mov ah, 4ch
       int 21h
codp ends
end start
```

Pentru a executa acest program el trebuie asamblat și linkeditat.

1) asamblarea fișierului **HELLO.ASM**

Vom scrie linia de comandă:

nume_asamblor HELLO

unde *nume_asamblor* este numele asamblorului pe care îl folosiți (de obicei TASM sau MASM). În urma asamblării se generează un fișier **HELLO.OBJ** care va conține codul obiect al programului.

2) linkeditarea fișierului **HELLO.OBJ**

Linia de comandă corespunzătoare este:

nume_linkeditor HELLO

unde *nume_linkeditor* este numele linkeditorului pe care îl folosiți (de obicei TLINK sau LINK). În urma linkeditării se generează un fișier **HELLO.EXE** care poate fi executat.

3) executarea fișierului **HELLO.EXE**

În urma execuției acestui program va fi afișat mesajul:

```
Hello World!
```

Să dăm câteva explicații relativ la programul de mai sus. La un moment dat microprocesorul poate avea acces la patru segmente logice:

- segmentul de stivă, accesibil prin registrul **SS**;
- segmentul de date curent, accesibil prin registrul **DS**;
- segmentul de cod, accesibil prin registrul **CS**;
- segmentul de date suplimentar, accesibil prin registrul **ES**.

Aceste segmente logice pot corespunde la patru segmente fizice distincte, dar pot exista și suprapuneri. În programul anterior sunt definite trei segmente, definițiile lor fiind delimitate de directivele **segment** și **ends**.

Primul segment este numit *stiva* și definește o stivă cu dimensiunea de 256 de bytes ($100_{16} = 256_{10}$). Pentru segmentul de stivă se recomandă utilizarea cuvântului cheie **stack** imediat după **segment** pentru a informa asamblorul că segmentul respectiv este un segment de stivă. Dacă se creează un segment de stivă fără a se utiliza cuvântul **stack**, atunci trebuie date instrucțiuni pentru inițializarea regiștrilor **SS** și **SP**. Dimensiunea stivei este declarată folosind directiva DB care permite declararea de date cu dimensiunea de câte un byte. Caracterul '?' folosit în această declarație indică faptul că zona de memorie trebuie doar rezervată, nu și inițializată. Operatorul dup stabilește numărul de zone de memorie contigue de aceeași mărime care se vor defini cu ajutorul directivei **DB**.

Cel de-al doilea segment (numit *date*) definește un segment de date în care este memorat un singur șir de caractere care reprezintă mesajul care va fi afișat pe ecran. Șirul de caractere se termină cu '$'. Acest caracter este precedat de caracterele cu codurile ASCII 13 și 10 (Carriage Return - CR și Line Feed - LF).

Directiva **assume** indică asamblorului segmentele asociate cu fiecare registru de segment, astfel **SS** va fi asociat cu segmentul de stivă, **DS** cu segmentul de date și **CS** cu segmentul de cod.

Ultimul segment (codp) conține instrucțiunile programului.

În continuare vă vom prezenta efectele fiecărei instrucțiuni în parte.

Primele două instrucțiuni au ca efect setarea registrului **DS** ca pointer la segmentul de date (registrul **DS** va conține adresa de bază a acestui segment).

Această setare nu poate fi făcută direct, deoarece instrucțiunea **mov** nu poate fi executată în legătură cu regiștrii de segment și, ca urmare, nu se poate scrie direct: `mov ds, date`. Din acest motiv adresa acestui segment este memorată mai întâi în registrul **AX** (registru acumulator) prin instrucțiunea: `mov ax, date`. Doar după aceea această valoare este copiată din registrul **AX** în registrul **DS** prin instrucțiunea `mov ds, ax`.

Următoarele instrucțiuni apelează două funcții DOS:

- funcția 9h trimite spre dispozitivul standard de ieșire un șir de caractere terminat cu '$', acest ultim caracter nefiind tipărit;
- funcția 4Ch permite terminarea unui program și cedarea controlului către procesul părinte (în acest caz sistemul de operare).

Pentru a apela o funcție DOS trebuie parcurși următorii pași:

- se încarcă numărul funcției în registrul **AH**;
- se încarcă, dacă este cazul, numărul subfuncției în registrul **AL**;
- se încarcă eventuale alte date în regiștrii corespunzători;
- se generează întreruperea 21h.

Instrucțiunea `mov dx, offset Mesaj` are ca efect încărcarea deplasamentului șirului față de baza segmentului în registrul **DX**.

Fișierul sursă se încheie cu directiva **end** care permite inițializarea regiștrilor **CS** și **IP** cu adresa specificată după cuvântul cheie **end**. Toate fișierele care conțin surse scrise în limbaj de asamblare ar trebui să se încheie cu această directivă.

Definiții de segment complete

Un segment este definit folosind directivele **segment** și **ends**, aceste directive marcând începutul, respectiv sfârșitul definiției segmentului. Sintaxa definiției unui segment este următoarea:

nume **segment** *[aliniere] [tip] [utilizare] ['clasa']*
corpul segmentului
nume **ends**

- *nume* reprezintă numele asociat segmentului și acesta nu poate fi utilizat cu nici o altă semnificație în program. El are asociată o valoare, reprezentată pe 16 biți, care reprezintă adresa de segment corespunzătoare poziției sale în memorie în timpul execuției. Un anumit segment poate fi deschis și închis de oricâte ori.

- *aliniere* poate avea una dintre valorile **byte**, **word**, **para**, **page** sau **mempage** și indică tipul adresei de început a segmentului. Mai precis, *aliniere* indică faptul că adresa de început a zonei de memorie rezervată segmentului este divizibilă cu 1, 2, 16, 256 sau 4096, implicit fiind `para`. Linkeditorul folosește informațiile de aliniere pentru a determina adresa de pornire relativă pentru fiecare segment.

- *tip* este o informație care se adresează linkeditorului, ea indică raportul dintre acest segment și segmentele definite în alte module obiect. Valorile pe care le poate lua sunt **public**, **private**, **memory**, **common**, **virtual**, **at** *expresie*, **stack** sau **uninit** (tipul implicit al unui segment este **private**):
  - În cazul în care tipul segmentului este **public** atunci dacă într-un alt modul obiect legat cu modulul obiect curent există un segment cu același nume, cele două segmente vor fi concatenate.
  - Dacă tipul segmentului este **private** atunci segmentul nu poate fi combinat cu alte segmente având același nume dar aflate în afara modulului în care a fost creat segmentul.
  - Pentru tipul **memory** segmentul va fi memorat în spațiul rămas disponibil după așezarea celorlalte segmente în memorie. Segmentele de tipul **memory** care au același nume sunt concatenate.
  - Dacă segmentul este de tip **common** atunci două sau mai multe segmente cu același nume aflate în module diferite vor fi suprapuse (vor începe la aceeași adresă) și lungimea zonei rezultate va fi egală cu lungimea celui mai mare segment.
  - Tipul **virtual** indică faptul că segmentul face parte dintr-o clasă specială de segment care trebuie să fie declarată inclusă în interiorul altui segment. Segmentele având acest tip sunt tratate de linkeditor ca fiind zone comune și sunt atașate segmentului în care sunt incluse.
  - Dacă tipul segmentului este **at** *expresie* atunci segmentul va fi încărcat în memorie la adresa obținută prin evaluarea expresiei.
  - Dacă un segment este declarat de tipul **stack**, segmentul respectiv este considerat ca fiind o stivă. Segmentele de tip **stack** care au același nume vor fi concatenate.
  - Declararea unui segment de tipul **uninit** produce un mesaj de avertizare în cazul în care există date inițializate, necorespunzătoare pentru segmente de date neinițializate.

- *utilizare* poate fi folosit numai începând cu microprocesorul 80386 și specifică dacă instrucțiunile operează asupra unor date reprezentate pe 16 biți sau asupra unor date reprezentate pe 32 de biți. Valorile lor pot fi `USE16` sau `USE32` în funcție de lungimea reprezentării dorite. Dacă o instrucțiune aflată într-un segment cu atributul `USE16` face referire la un operand pe 32 de biți atunci instrucțiunea va fi prefixată (în momentul asamblării) cu un prefix având valoarea `66h`, prefix ce indică inversarea valorii implicite pentru dimensiunea operanzilor. Dacă se folosește un mod de adresare pe 32 de biți atunci asamblorul va prefixa instrucțiunea cu prefixul `67h`, prefix ce indică inversarea valorii implicite pentru dimensiunea unui *AE*. Același lucru se întâmplă și în cazul în care o instrucțiune aflată într-un segment cu atributul `USE32` folosește un mod de adresare pe 16 biți. În modul real de funcționare (folosit de sistemul de operare MS-DOS) toate segmentele de cod trebuie să aibă atributul `USE16` pentru a putea fi executate. În mod protejat pot fi folosite ambele tipuri de segmente.

- *clasa* reprezintă un nume cuprins între apostrofuri și are rolul de a controla ordinea în care sunt păstrate în memorie segmentele care compun un program. Astfel, pot fi asociate segmente cu nume diferite, dar care au "scopuri" similare. Dacă un segment are clasa neprecizată atunci clasa sa se consideră a fi șirul vid. Segmentele având aceeași clasă vor fi memorate la adrese succesive. Un alt factor care controlează ordinea segmentelor în memorie este ordinea segmentelor din fișierul sursă. Directiva **dosseg** înlocuiește toți ceilalți factori în determinarea ordinii segmentelor; ea indică faptul că segmentele vor fi ordonate respectând convenția de ordonare a segmentelor DOS.

Directiva end

Directiva **end** poate fi folosită pentru marcarea sfârșitului unui fișier sursă în limbaj de asamblare și pentru a inițializa regiștrii **CS** și **IP**. Sintaxa acestei directive este:

end *[adresă_început]*

unde *adresă_început* este o etichetă sau o expresie care identifică adresa de la care programatorul dorește începerea execuției atunci când programul este încărcat. Orice fișier sursă în limbaj de asamblare trebuie să se încheie cu această directivă, în caz contrar la asamblare apare o eroare de tipul:

```
Unexpected end of file encountered.
```

Dacă programul conține mai multe module, numai unul dintre ele (de obicei cel ce conține programul principal) va defini o *adresă_început*. Dacă această adresă nu este precizată, se presupune că ea nu există și execuția programului va începe de la o adresă greșită (întâmplătoare). Asamblorul și linkeditorul nu vor genera nici o eroare în acest caz.

Încărcări și memorări

Microprocesoarele din familia 8086 pun la dispoziție mai multe instrucțiuni pentru încărcarea, memorarea și transferul diferitelor date. Cea mai importantă instrucțiune de acest tip este instrucțiunea **mov** și are sintaxa:

mov *destinație, sursă*

După execuția acestei instrucțiuni operandul sursă și operandul destinație vor conține aceeași valoare și anume cea care se afla în operandul sursă înaintea execuției instrucțiunii. Operandul destinație poate fi un registru sau o locație de memorie. Așa cum am mai arătat, unele transferuri necesită două instrucțiuni **mov**, fiind nevoie de un registru intermediar pentru ca transferul să poată avea loc. Instrucțiunile următoare ilustrează tipurile de transfer care pot fi realizate cu o singură instrucțiune **mov**:

```
mov ax, 12        - adresare imediată → operanzi registru
mov mem, 12       - adresare imediată → adresare directă
mov mem[bx],12 - adresare imediată → adresare indirectă
mov mem, ds       - registru de segment → memorie
mov mem, ax       - registru → adresare directă
mov ax, mem       - adresare directă → registru
mov ax, mem[bx]- adresare indirectă → registru
mov ds, mem       - memorie → registru de segment
mov ax, bx        - registru → registru
mov ds, ax        - registru general → registru de segment
mov ax, ds        - registru de segment în registru general
```

Vă prezentăm în continuare câteva tipuri de transferuri care necesită două instrucțiuni **mov**.

Transfer imediat în registru de segment:

```
mov ax, @data  - încărcare imediată la registru general
mov ds, ax - stocare registru general în registru segment
```

Transfer din memorie în memorie:

```
mov ax, mem1   - încărcare memorie la registrul general
mov mem2, ax   - stocare registru general în memorie
```

Transfer din registru segment în registru segment:

```
mov ax, ds - încărcare registru segment la registru general
mov es, ax - stocare registru general în registru segment
```

O altă instrucțiune importantă este instrucțiunea **xchg** care are sintaxa:

xchg *destinație, sursă*

și are ca efect interschimbarea conținutului sursei și al destinației. Pot fi schimbate date între regiștri sau între regiștri și memorie.

Instrucțiuni aritmetice

Există mai multe instrucțiuni care permit efectuarea de operații aritmetice, cele mai importante fiind: **add**, **inc**, **sub**, **dec**, **neg**, **mul**, **imul**, **div** și **idiv**.

Instrucțiunea **add** are sintaxa:

add *destinație, sursă*

și are ca efect însumarea operanzilor sursă și destinație și stocarea rezultatului în operandul *destinație*, vechea valoare a acestui operand fiind înlocuită cu noua valoare. Cei doi operanzi pot fi numere binare, cu sau fără semn, reprezentate pe 8 sau 16 biți (sau 32 de biți începând cu microprocesorul 80386), ambii trebuind să aibă aceeași dimensiune. Cei doi operanzi pot fi operanzi registru sau operanzi în memorie (nu ambii, deoarece operațiile *memorie-la-memorie* nu sunt admise niciodată). *Sursa* poate fi și un operand imediat. Instrucțiunea **add** actualizează indicatorii AF, CF, OF, PF, SF și ZF.

Instrucțiunea **inc** are sintaxa:

inc *destinație*

și are același efect ca instrucțiunea:

add *destinatie*, 1

adică adaugă 1 la operandul *destinație* și depune rezultatul tot în operandul *destinație*, fiind înlocuită vechea valoare. Operandul *destinație* poate fi un număr binar fără semn de 8, 16 sau 32 (începând doar cu 80386) de biți și poate fi un operand registru sau un operand în memorie. În urma execuției acestei instrucțiuni sunt actualizați indicatorii AF, OF, PF, SF și ZF.

Instrucțiunea **sub** are sintaxa:

sub *destinație, sursă*

și are ca efect scăderea operandului sursă din operandul *destinație* și înlocuirea valorii operandului *destinație* cu rezultatul scăderii. Operanzii pot fi numere binare de 8, 16 sau 32 (80386) de biți, ambii trebuind să aibă aceeași dimensiune. Cei doi operanzi pot fi operanzi registru sau operanzi în memorie (nu amândoi). *Sursa* poate fi și un operand *imediat*. Instrucțiunea **sub** actualizează indicatorii AF, CF, OF, PF, SF și ZF.

Instrucțiunea **dec** are sintaxa:

dec *destinație*

și are același efect ca instrucțiunea:

sub *destinație*, 1

adică scade 1 din valoarea operandului destinație și depune rezultatul tot în operandul destinație, fiind înlocuită vechea valoare. Operandul destinație poate fi un număr binar fără semn de 8, 16 sau 32 de biți (începând doar cu 80386) și poate fi un operand registru sau un operand în memorie. În urma execuției acestei instrucțiuni sunt actualizați indicatorii AF, OF, PF, SF și ZF.

Instrucțiunea **neg** are sintaxa:

neg *destinație*

și are ca efect scăderea din 0 a operandului *destinație* și depunerea rezultatului tot în operandul *destinație*. Instrucțiunea este echivalentă cu secvența următoare (cu condiția ca operandul *destinație* să nu fie exact registrul **AX**; dacă această condiție nu e îndeplinită putem folosi alt registru):

```
mov ax, destinație
mov destinație, 0
sub destinație, ax
```

În urma execuției instrucțiunii sunt actualizați indicatorii AF, CF, OF, PF, SF și ZF.

Instrucțiunea **mul** are sintaxa:

mul *sursă*

și efectuează o înmulțire fără semn. Operandul *sursă* este înmulțit cu registrul acumulator (**AL** dacă *sursa* are o reprezentare pe 8 biți, **AX** pentru o reprezentare pe 16 biți și, eventual, **EAX** pentru o reprezentare pe 32 de biți). În funcție de lungimea reprezentării sursei, rezultatul este stocat fie în registrul **AX**, fie în regiștrii **DX** și **AX**, fie în regiștrii **EDX** și **EAX**. În urma execuției instrucțiunii sunt actualizați indicatorii CF și OF.

Instrucțiunea **imul** are sintaxa:

imul *sursă*

și efectuează o înmulțire cu semn, aceasta fiind singura diferență față de instrucțiunea **mul**. Începând cu microprocesorul 80186 instrucțiunea **imul** are două sintaxe suplimentare:

imul *registru, operand_imediat*
imul *registru, operand_în_memorie, operand_imediat*

În această situație cei doi operanzi care se înmulțesc sunt considerați pe 16 biți și destinația este întotdeauna un registru. Începând cu microprocesorul 80386 aceste instrucțiuni pot fi extinse la 32 de biți. În situația în care sunt specificați doi operanzi, se înmulțește operandul imediat cu registrul, iar rezultatul este stocat în registru. În situația în care sunt specificați trei operanzi se înmulțesc operandul imediat cu operandul în memorie și rezultatul este stocat în registru. Începând cu microprocesorul 80386 apare încă o sintaxă suplimentară și anume:

imul *registru, sursă*

În acest caz operandul *sursă* poate fi atât un registru, cât și o variabilă din memorie de aceeași dimensiune cu destinația. În urma execuției instrucțiunii sunt actualizați indicatorii CF și OF.

Instrucțiunea **div** are sintaxa:

div *sursă*

și efectuează o înmulțire fără semn. Registrul acumulator (și eventual extensia lui) este împărțit la operandul *sursă*. În funcție de lungimea reprezentării sursei, câtul este stocat fie în registrul **AL**, fie în registrul **AX**, fie în registrul **EDX**, iar restul în regiștrii **AH**, **DX** sau **AF**. Dacă valoarea restului nu poate fi reprezentată în registrul destinație sau dacă se încearcă o împărțire la 0, se generează o întrerupere de ordin 0.

Instrucțiunea **idiv** are sintaxa:

idiv *sursă*

și efectuează o împărțire cu semn, aceasta fiind singura diferență față de instrucțiunea **div**.

Comparări și salturi

Cea mai simplă și comodă modalitate de a transfera controlul în limbaj de asamblare este folosirea unor instrucțiuni de salt condiționat. Un astfel de salt condiționat are loc în doi pași: se testează o anumită condiție (printr-o instrucțiune care poziționează anumiți indicatori) și, în cazul în care condiția este adevărată, se efectuează saltul; în caz contrar se continuă programul cu instrucțiunea următoare.

Pentru efectuarea unei comparări, cea mai utilizată instrucțiune este **cmp**, instrucțiune ce are sintaxa:

cmp *destinație, sursă*

Această instrucțiune are aproape același efect ca și instrucțiunea **sub**, singura deosebire fiind faptul că rezultatul scăderii nu mai este stocat în operandul *destinație*, valorile vechi ale celor doi operatori rămânând neschimbate. În urma acestei instrucțiuni sunt actualizați indicatorii AF, CF, OF, PF, SF și ZF.

Instrucțiunile de salt operează asupra registrului de segment **CS** și a pointerului de instrucțiuni **IP**. Datorită faptului că acești regiștri sunt modificați, programul nu se va mai executa în ordinea secvențială, următoarea instrucțiune neaflându-se în locația de memorie imediat următoare locației în care se află instrucțiunea care a determinat saltul. Acest fapt determină UIM să încarce alte instrucțiuni folosind noile valori ale regiștrilor **CS** și **IP**.

Instrucțiunile de salt condiționat execută un salt la o instrucțiune în funcție de anumite stări ale indicatorilor. Dacă o anumită condiție este îndeplinită, atunci va avea loc un salt la o instrucțiune specificată. Sintaxa unei instrucțiuni de salt condiționat este:

mnemonică etichetă

unde *mnemonică* este un cod pentru condiția testată, iar *eticheta* indică eticheta instrucțiunii la care se face saltul. Tabelul 1 prezintă lista ***mnemonicelor*** instrucțiunilor de salt condiționat împreună cu condițiile care se testează în fiecare caz.

Se observă faptul că, în unele cazuri, saltul se produce în cazul în care condiția este îndeplinită și, în alte cazuri, acesta are loc numai dacă această condiție nu este îndeplinită. Există și extensii ale acestor mnemonice pentru microprocesoarele 80386 și cele ulterioare cum ar fi, de exemplu, mnemonica **jecxz** care testează dacă valoarea din registrul **ECX** este 0.

Literele din care este formată mnemonica au fiecare o anumită semnificație care poate fi văzută în tabelul 2 pentru cele mai importante litere.

Numele mnemonicelor instrucțiunilor de salt se referă întotdeauna la relația pe care primul operand al instrucțiunii **cmp** anterioare o are cu cel de-al doilea operand al

acestei instrucțiuni. Se observă că anumite instrucțiuni de salt condiționat au două mnemonice, caz în care poate fi folosită oricare dintre ele. Instrucțiunile de salt testează starea unuia sau mai multor indicatori în cadrul condiției. Există o singură excepție de la această regulă, și anume instrucțiunea **jcxz** care execută un salt bazat pe valoarea registrului **CX**.

Instrucțiunile de salt necondiționat transferă controlul la o anumită instrucțiune, fără a mai fi testată nici o condiție. Sintaxa instrucțiunii de salt necondiționat este următoarea:

jmp *destinație*

Cu ajutorul instrucțiunilor de comparare și de salt pot fi simulate structurile de decizie de tipul **if-then-else.**

Structurile repetitive pot fi simulate utilizând instrucțiunile de salt condiționat, dar microprocesoarele din familia 8086 oferă o serie de instrucțiuni suplimentare destinate creării de bucle repetitive. Sintaxa acestor instrucțiuni este identică cu cea a instrucțiunilor de salt condiționat și anume:

mnemonică etichetă

Saltul se efectuează la instrucțiunea etichetată cu *etichetă* în cazul în care registrul **CX** este diferit de zero. La fiecare iterație acest registru este decrementat cu o unitate, deci, registrul **CX** este folosit drept contor pentru numărul de iterații al buclei.

Dacă este folosită mnemonica **loop** se execută bucla atât timp cât valoarea din registrul **CX** nu este 0. Mnemonicele **loope** și **loopz** sunt folosite în bucle care verifică o anumită condiție, sensul lor fiind: **loop while equal**. Mnemonicele **loopne** și **loopny** sunt folosite tot în buclele ce verifică o condiție, însă sensul lor este: **loop while not equal.**

Noțiunile prezentate anterior sunt utile pentru buna înțelegere a modalității în care funcționează un microprocesor și vor fi de un real folos în cazul în care se dorește studierea în detaliu a programării în limbaj de asamblare.

Tabel 1

Instrucțiunile de salt condiționat

Nr.	Mnemonică	Condiția care este testată	Explicarea condiției
1	JB	CF=1	este mai mic
2	JNAE	CF=1	nu este mai mare și nici egal
3	JL	(SF **xor** OF)=1	este mai mic
4	JNGE	(SF **xor** OF)=1	nu este mai mare
5	JBE	(CF **or** ZF)=1	este mai mic sau egal
6	JNA	(CF **or** ZF)=1	nu este mai mare
7	JLE	((SF **xor** OF)**or** ZF)=1	este mai mic sau egal
8	JNG	((SF **xor** OF)**or** ZF)=1	nu este mai mare
9	JE	ZF=1	este egal
10	JZ	ZF=1	este zero
11	JNE	ZF=0	nu este egal
12	JNZ	ZF=0	nu este zero
13	JAE	CF=0	este mai mare sau egal
14	JNB	CF=0	nu este mai mic
15	JGE	(SF **xor** OF)=0	este mai mare sau egal
16	JNL	(SF **xor** OF)=0	nu este mai mic și nici egal
17	JA	(CF **or** ZF)=0	este mai mare
18	JNBE	(CF **or** ZF)=0	nu este mai mic și nici egal
19	JG	((SF **xor** OF)**or** ZF)=0	este mai mare
20	JNLE	((SF **xor** OF)**or** ZF)=0	nu este mai mic și nici egal
21	JNS	SF=0	nu are semn
22	JS	SF=1	are semn
23	JNO	OF=0	nu a apărut o depășire
24	JO	OF=1	a apărut o depășire
25	JNC	CF=0	nu a apărut un transport
26	JC	CF=1	a apărut un transport
27	JNP	PF=0	nu este paritate
28	JPO	PF=0	este paritate impară
29	JP	PF=1	este paritate
30	JPE	PF=1	este paritate egală
31	JCXZ	CX=0	valoarea registrului CX este 0

Bibliografie:

Octavian Aspru - *Elemente de limbaj de asamblare*, Editura ADIAS, 1996

Tabel 2

Semnificația literelor

Litera	Proveniența	Semnificația
J	**Jump**	Salt
L	**Less**	Mai mic decât (pentru comparații fără semn)
E	**Equal**	Egal
G	**Greater**	Mai mare decât (pentru comparații fără semn)
B	**Below**	Dedesubt (pentru comparații cu semn)
A	**Above**	Deasupra (pentru comparații cu semn)
N	**Not**	Nu

Programarea PLĂCII de SUNET

Adrian Monea

După ce în episoadele precedente am studiat posibilitățile populare plăci de sunet SoundBlaster, în acest al treilea (și ultim) episod vom prezenta îmbunătățirile aduse de placa SoundBlaster Pro. De asemenea vom prezenta formatul WAV, un format foarte răspândit în lumea Windows.

Generarea sunetelor stereo FM pe SoundBlaster Pro se realizează la fel ca pe SoundBlaster, însă se utilizează porturi diferite pentru canalele stâng și drept.

Generarea sunetelor *stereo* folosind DSP-ul se realizează la fel ca în cazul metodei *mono*, însă trebuie trimiși doi octeți pentru fiecare eșantion. Primul este trimis în canalul stâng, iar al doilea în cel drept. De asemenea, chip-ul de mixer trebuie resetat, iar placa de sunet trebuie "informată" că se dorește redarea unui sunet stereo. Acest aspect prezintă un avantaj evident, și anume acela că se poate stoca informația pentru ambele canale într-un singur bloc de date, iar acesta se poate transfera folosind un singur canal DMA. Formatul WAV chiar stochează sunetul în acest mod.

Octeții pentru canalul stâng

0 1 2 3 4 ...

0 1 2 3 4

Octeții pentru canalul drept

Pentru a reda sunetul SoundBlaster Pro este setat pentru redare stereo și chipul DMA este programat pentru a trimite datele exact așa cum sunt stocate.

Chipul de mixer CT 1345

Pentru a accesa regiștrii mixerului se procedează în același mod ca în cazul regiștrilor normali SoundBlaster, dar portul $2x4 este portul de index, iar $2x5 este portul de citire/scriere date, unde

x=2 pentru adresa de bază setată la $220,
x=3 pentru adresa de bază setată la $230,
x=4 pentru adresa de bază setată la $240.

Setarea unui registru de mixer pe o valoare dată poate fi realizată de următoarea procedură:

```
{ base = 220h, 230 sau 240h }
procedure SetMixer(base:Word; index,val:Byte);
begin
  Port[base+4]:=index;
  Port[base+5]:=value
end;
```

De asemenea, citirea valorii curente a unui registru poate fi realizată astfel:

```
function GetMixer(base:Word;index:Byte):Byte;
begin
  Port[base+4]:=index;
  GetMixer:=Port[base+5]
end;
```

Registrul *Data Reset* este folosit pentru a reseta chipul de mixer. Registrul trebuie setat pe 0 înainte de a schimba orice alți regiștri ai mixerului.

Registrul *Input* selectează sursa de intrare și tipul de filtru.

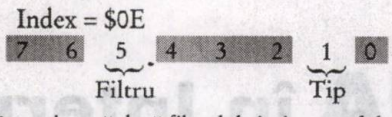
Index = $0C

7	6	5	4	3	2	1	0
		Filtru			Sursă		

Biții 3-5 precizează tipul filtrului:
000 - Low
001 - High
010 - Fără filtru

Biții 1-2 precizează sursa:
00 - Microfon 1
01 - CD
10 - Microfon 2
11 - Line In

Registrul *Output* determină dacă sunetul de intrare este *stereo* sau *mono*. Pentru stereo trebuie trimiși doi octeți pentru fiecare eșantion. Acest registru permite și ignorarea filtrului de ieșire.

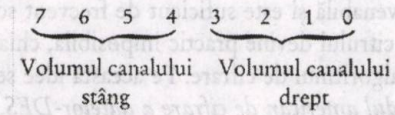


Bitul 5 precizează dacă filtrul de ieșire este folosit sau nu:
0 - folosește filtrul sau 1 - ignoră filtrul
Bitul 1 precizează tipul sunetului:
0 - Mono sau 1 - Stereo

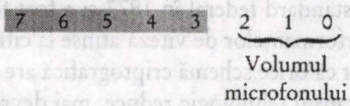
În continuare este prezentat un tabel cu o serie de regiștri ce permit setarea separată a volumului pentru fiecare din cele două canale.

Index	Semnificație
$22	Master Volume - permite setarea volumului master.
$04	Voice Volume - permite setarea volumului pentru redarea DSP.
$26	FM Volume - permite setarea volumului pentru sinteza FM.
$28	CD Volume - permite setarea volumului pentru ieșirea CD.
$2E	Line In Volume - permite setarea volumului pentru intrarea directă (Line In).

Octeții de la adresele de mai sus au următoarea structură:



Pentru setarea volumului se folosește și registrul $0A (*Mic Mixing*), dar acesta are următoarea structură:



Formatul WAV

Fișierele Wav folosesc structura de fișier RIFF. Formatul RIFF a fost proiectat special pentru scopuri multimedia. Un fișier RIFF este format din mai multe "bucăți" (tabelul 1).

Tabelul 1.

Structura unui fișier RIFF

Deplasare	Denumire	Lungime (în octeți)	Descriere
$00	rID	$4	Conține caracterele "RIFF"
$04	rLen	$4	Lungimea datelor următoarei bucăți.
$08	rData	rLen	Bucata de date.

Secțiunea de date *rData* dintr-un fișier WAV este împărțită mai departe în mai multe subsecțiuni, așa cum se observă din tabelul 2.

Tabelul 2.

Structura bucății rData

Deplasare în rData	Nume	Lungime (în octeți)	Descriere
$00	wID	4	Conține caracterele "WAVE"
$04	Format	20	Conține date ce specifică formatul datelor care urmează
$18	Date	?	Conține datele audio WAV

Structura subsecțiunii *Format* din secțiunea *rData* este alcătuită din câmpurile prezentate în tabelul 3.

Tabelul 3.

Structura bucății Format

Deplasare	Nume	Lungime (în octeți)	Descriere
$00	fID	4	Conține caracterele "fmt"
$04	fLen	4	Lungimea datelor
$08	wFormatTag	2	*
$0A	nChannels	2	Numărul de canale: 1-mono, 2-stereo
$0C	nSamplesPerSec	2	Frecvența de redare
$0E	nAvgBytesPerSec	2	**
$10	nBlockAlign	2	***
$12	FormatSpecific	2	Zona specifică

* wFormatTag specifică formatul undelor, de ex. 1=Pulse Code Modulation (sau altfel spus sunet normal eșantionat pe 8 biți și necomprimat)
** Indică numărul mediu de octeți pe secundă la care ar trebui transferate datele și este egal cu:
= nChannels * nSamplesPerSec * (nBitsPerSample/8)
*** Indică aliniamentul blocului de date din bucata de date. Programele trebuie să proceseze un multiplu de nBlockAlign o dată.
nBlockAlign = nChannels * (nBitsPerSample/8)

Secțiunea de date este împărțită în următoarele câmpuri:

Deplasare	Nume	Lungime (în octeți)	Descriere
$00	dID	4	Conține caracterele "data"
$04	dLen	4	Lungimea datelor din câmpul dData
$08	dData	dLen	Datele propriu zise.

În fișierele mono pe 8 biți fiecare octet reprezintă un eșantion. În fișierele stereo pe 8 biți sunt stocați doi octeți pentru fiecare eșantion, primul fiind valoarea canalului stâng, iar al doilea cea a canalului drept.

Bibliografie:

1. Mark Feldman - *PCGPE (PC Games Programming Encyclopedia, format electronic)*
2. *Creative Labs Corporation - Sound Blaster Resource Kit*
3. *Sound Blaster - The Official Book , Osborne/McGraw Hill*

Adrian Monea este student la Universitatea Tehnică Cluj, Facultatea de Automatică și Calculatoare. Poate fi contactat prin intermediul redacției.

CRIPTOGRAFIA în Internet

Victor-Valeriu Patriciu

Puține sunt persoanele care să nu fi încercat, măcar o dată, transmiterea cifrată a unor mesaje personale "foarte importante". Și mai puține sunt, probabil, cele care nu s-au amuzat niciodată rezolvând jocuri REBUS unde, pe baza unor frânturi de mesaj, trebuie descoperită regula de reconstituire a întregului. Explicația este atracția firească pe care o prezintă un astfel de exercițiu intelectual și satisfacția pe care ți-o dă reușita rezolvării.

Internet, securitate informatică, criptografie

Criptografia sau *știința codificării informației* în vederea împiedicării acceselor neautorizate, are o lungă istorie, confidențialitatea comunicării fiind o cerință a tuturor timpurilor. Dacă ar trebui să alegem un singur exemplu al criptografiei "clasice", acesta ar fi ***cifrul lui Cezar***, nu atât datorită celebrității împăratului roman de care se leagă folosirea lui, ci pentru că principiul său de bază s-a menținut nealterat aproape două milenii. Dar iată care este acest principiu! Cifrarea este bazată pe o substituție a fiecărui simbol al mesajului cu un altul, aflat în alfabet la trei poziții după el. În felul acesta, celebrele cuvinte *VENI VIDI VICI* transmise de *Cezar* lui *Matius*, după victoria de la Zela asupra lui Farnace (anul 47 î.e.n.), ar putea fi codificate prin *YHQL YLGL YLFL*. Pentru a putea descifra mesajul, procedeul de cifrare se inversează, fiecare literă fiind înlocuită cu una aflată cu trei poziții înaintea sa în alfabet.

Slăbiciunea acestui cifru este îndeobște cunoscută, un criptoanalist modest reușind destul de repede să descopere atât *algoritmul* (subtituție circulară) cât și *cheia* (numărul de poziții de decalare-trei). Mai mult, odată descoperite aceste elemente, se poate deduce ușor și perechea algoritm-cheie folosită la cifrare, cifrarea fiind simetrică. Ca urmare, un intrus poate nu numai să violeze confidențialitatea unor mesaje, dar și să introducă în comunicație mesaje false. Multă vreme eforturile criptografilor au fost dirijate spre întărirea cifrurilor prin complicarea algoritmului, combinând substituții și transpoziții asupra unor simboluri sau asupra unor blocuri (grupe de simboluri). Istoria modernă a criptografiei cunoaște numeroase inovații în această privință. Elementele ce au marcat însă cotitura semnificativă în dezvoltarea metodelor criptografice sunt următoarele:

Dezvoltarea Internet-ului a fost un stimulent extraordinar, prin presiunea exercitată de tot mai mulți utilizatori, a căror dorință obiectivă este păstrarea secretului și a siguranței asupra poștei electronice private, a transferului electronic de documente și de fonduri și a altor aplicații. De asemenea, s-a constatat o potențare a gamei de instrumente folosite efectiv în execuția algoritmilor de cifrare. Utilizarea calculatoarelor electronice a permis folosirea unor chei de dimensiuni mai mari, sporindu-se astfel rezistența la atacuri criptoanalitice. Când cheia secretă are o dimensiune convenabilă și este suficient de frecvent schimbată, spargerea cifrului devine practic imposibilă, chiar dacă se cunoaște algoritmul de cifrare. Pe această idee se bazează și *standardul american de cifrare a datelor-DES*, larg utilizat de guvernul SUA și de diverse companii internaționale. Propus într-o formă inițială de IBM în 1975, DES a fost adoptat ca standard federal în 1977 și a fost folosit intens datorită performanțelor de viteză atinse la cifrare. Experiența a arătat că orice schemă criptografică are o viață limitată și că avansul tehnologic reduce, mai devreme sau mai târziu, securitatea furnizată de ea. Actualmente se lucrează la ***NIST- Institutul Național de Standarde și Tehnologie*** al SUA la selectarea unuia din cei 12 algoritmi candidați pentru noul standard criptografic evoluat - ***AES, Advanced Encryption System***, cerut de noile performanțe tehnologice atinse în lumea calculatoarelor de azi.

Un alt moment important în evoluția criptografiei moderne l-a constituit adoptarea unui principiu diferit de cel al cifrării simetrice. *Whitfield Diffie* și *Martin Hellman*, cercetători la Universitatea Stanford din California, au pus bazele ***criptografiei asimetrice cu chei publice*** în anul 1976.

În locul unei singure chei secrete, criptografia asimetrică folosește două chei diferite, una pentru cifrare, alta pentru descifrare. Deoarece este imposibilă deducerea unei chei din cealaltă, una dintre chei este făcută *publică*, fiind pusă la îndemâna oricui dorește să transmită un mesaj cifrat. Doar destinatarul, care deține cea de-a doua cheie, poate descifra și utiliza mesajul. Tehnica cheilor publice poate fi

folosită și pentru *autentificarea* mesajelor, fapt care i-a sporit popularitatea. Nu este deci de mirare că guvernul SUA a inițiat adoptarea unui standard de semnătură digitală bazat pe conceptul de cheie publică. Acest demers a generat controverse soldate chiar cu acuze între organizațiile implicate. Până în decembrie 1990, *NIST* recomanda pentru adoptare ca standard metoda RSA, prezentă deja în industrie. Nouă luni mai târziu, în august 1991, *NIST* a avansat un cu totul alt algoritm, bazat pe o metodă cu chei publice publicată de *Taher El Gamal* în 1985. Noua propunere, denumită ***DSS*** (***Digital Signature Standard***), a fost dezvoltată de Agenția de Securitate Națională a SUA.

Un moment remarcabil în evoluția criptografiei ultimilor ani îl constituie apariția conceptului de ***criptografie cu chei în custodie (Key Escrow Systems).*** El își are punctul de plecare în SUA anilor '90, unde s-a încercat impunerea, de către agențiile statului însărcinate cu securitatea națională, a unui sistem prin care acestea, sub aprobare legală, să poată decripta transmisiile în rețele ale unor firme sau persoane. Conceptul reprezintă nucleul central al *programului guvernamental Clipper*, numit și *Escrowed Encryption Standard* (***EES***), integrat unuia mai larg, numit *Capstone*. În decursul ultimilor 20 de ani s-a acumulat o anumită tensiune privind interesele publice pentru protecția informațiilor și interesele guvernamentale pentru obținerea accesului la informațiile adversarilor. Programul *Clipper Key Escrow* a fost o încercare de depășire a acestei tensiuni, oferind publicului o criptografie bună, păstrându-se pentru organele de impunere legală abilitatea de a decripta comunicațiile, când acest lucru este autorizat. În același timp, atât utilizatorii singulari cât și organizațiile se tem de consecințele pierderii cheilor. Un sistem care ar putea satisface atât cerințele utilizatorilor singulari și a corporațiilor, dar care să îndeplinească și cerințele guvernamentale de decriptare autorizată, ar putea rezolva această problemă de interes internațional. Asigurarea confidențialității comunicațiilor și a datelor memorate în fișiere sau baze de date devine rapid parte integrantă a infrastructurii noastre informaționale. Scopul explicit al criptografiei este de a face dificil sau imposibil, pentru o terță parte, accesul la informația protejată. În mod clar, devine o problemă cazul în care această terță parte este o autoritate a statului, care, din motive legale sau sociale, consideră uneori că are dreptul de acces la această informație. Motivațiile legale sau sociale ale acestui "drept de ascultare" sunt discutabile și au fost îndelung dezbătute. S-a ajuns la concluzia că avem nevoie de mecanisme care să permită accesul la informație a unei terțe persoane autorizate, continuând însă să se protejeze informația în fața altor persoane. Se folosește termenul de *criptografie cu chei în custodie* pentru a desemna asemenea sisteme.

Răspândirea explozivă a rețelelor de calculatoare și în primul rând a Internet-ului, a accentuat problema securității acestora și a evidențiat, odată în plus, rolul criptografiei în asigurarea ei. Demonstrația cea mai răsunătoare în acest sens s-a făcut în noiembrie 1988, când *Robert Morris* - doctorand la Universitatea Cornell - a răspândit panică în lumea universitară. Violând anumite reguli ale rețelei de calculatoare Internet (reguli ce alcătuiesc un soi de "gentlmen's agreement"), autorul atacului a reușit să infecteze printr-un program cu auto-reproducere, mii de calculatoare VAX și Sun 3, funcționând sub o anumită versiune de sistem de operare Unix. Incidentul a avut efecte majore în lumea calculatoarelor, înființarea unei echipe de consultanță în cazuri urgente (***CERT*** - *Computer Emergency Response Team*) pentru Internet și mărirea fondurilor alocate de firme pentru probleme de securitate a sistemelor de calcul fiind doar două dintre cele mai importante. Deși nu discută amănunte despre breșele din sistemele de securitate, soluțiile multor probleme importante sunt date de specialiștii CERT sub forma unor recomandări.

Internet-ul reprezintă cel mai mare proiect informatic de interconectare umană creat vreodată pe pământ. Evoluția sa este de-a dreptul spectaculoasă: de la 4 calculatoare interconectate în 1969, la 2800 în 1987, 535.000 în 1991, 1.7 milioane în 1993, 3.2 milioane în 1994 și aproape 10 milioane în 1998, cuprinzând astăzi câteva sute de mii de rețele conectate. În fiecare moment câteva zeci de milioane de utilizatori sunt conectați la Internet. ***Societatea modernă informatizată (satul global)*** reprezintă deja o realitate în care se ignoră frontierele și se trece peste orice constrângeri de ordin spațial sau temporal. Economia, politica și societatea se bazează astăzi din ce în ce mai mult pe această infrastructură informatică. De asemenea, guvernele, firmele din sectorul public și privat, organismele financiare naționale și internaționale, învățământul, cultura și cercetarea științifică, toate beneficiază de aceste forme eficiente de conducere, informare și comunicare.

O serie de considerente, care privesc utilizarea informaticii în toate aceste domenii, fac ca ***securitatea informatică*** să fie considerată în momentul de față *vitală pentru evoluția societății*:

- ***Dependența puternică de informatică.*** Agențiile guvernamentale, firmele, băncile etc. se văd în imposibilitatea de a lucra fără a dispune în mod sigur de programele și datele memorate în rețelele lor de calculatoare.
- ***Vulnerabilitatea tehnologiilor informatice.*** Se referă la faptul că organizațiile și firmele nu pot avea încredere totală în sistemele informatice folosite și în măsurile de protecție efectivă. Sistemele sunt vulnerabile la penetrări neautorizate, la distrugeri și modificări accidentale sau voite de date sau programe. Aceste sisteme pot deservi elemente vitale pentru societate cum ar fi sisteme militare, bănci, spitale, sisteme de transport, burse de valori, oferind în același timp un cadru de comportament antisocial sau de terorism. Tendința actuală privind amplificarea conectivității, în special în Internet, sporește aceste vulnerabilități; este din ce în ce mai greu să se localizeze un punct de acces ilegal în rețea, un utilizator având comportament inadecvat. Mult timp securitatea s-a ba-

zat pe respectul reciproc al utilizatorilor, pe "gentlemen agreement-ul" stabilit între aceştia şi mai puţin pe măsuri tehnice şi administrative. Odată însă ce Internet-ul s-a extins, comunitatea utilizatorilor a crescut foarte mult. Între aceştia se găsesc numeroase persoane care nu mai respectă regulile de comportament stabilite iniţial de cercetători. Vulnerabilitatea sistemelor informatice actuale poate antrena pierderi imense de ordin financiar direct sau indirect, prin "scurgerea" de informaţii confidenţiale având caracter personal, economic sau militar.

- ***Comerţul electronic.*** Datorită acestuia, în ultimii doi ani am asistat la marea "explozie" a Internet-ului. Acesta va face să cadă toate barierele fizice existente azi în calea comerţului, creând posibilitatea ca şi cele mai mici afaceri să acceseze o piaţă extinsă la întreaga lume. În acelaşi timp, clienţii pot face afaceri şi cumpărături de la firme care altfel nu le erau accesibile. *Asigurarea securităţii comerţului Internet* reprezintă însă cea mai importantă provocare ce stă în faţa profesioniştilor informaticieni. Dacă în urmă cu trei-patru ani comerţul Internet nici nu exista, astăzi acesta reprezintă un loc de atragere a unor enorme interese financiare; ca urmare tot mai multe firme de calculatoare se întrec în a oferi echipamentele şi programele necesare acestor servicii. Internet-ul devine foarte atractiv pentru micile firme, care pot fi foarte active pe reţea, se pot face mult mai rapid cunoscute şi îşi pot promova mai ieftin politica de piaţă. În acest context, multe afaceri au migrat în ultimii ani pe Internet. Practic, nu există firmă într-o ţară cu o dezvoltare cel puţin medie care să nu aibă pagini Web, pe care se găsesc informaţii despre serviciile şi produsele oferite. De asemenea, consumatorii şi producătorii pot comunica instantaneu prin e-mail, ceea ce le oferă posibilitatea unei informări reciproce şi a unor comunicaţii foarte ieftine. Şi totuşi aceştia nu s-au "aruncat" încă să facă comerţ la scară mare pe Internet. De ce oare această reţinere? Motivul este securitatea tranzacţiilor comerciale.

În aceste circumstanţe, ***securitatea informatică*** a devenit una din componentele majore ale Internet-ului. Utilizarea serviciilor de poştă electronică, Web, transfer electronic de fonduri şi documente etc., se bazează pe un sentiment adeseori fals, de securitate a comunicaţiilor, care poate transforma potenţialele câştiguri generate de accesul rapid la informaţii, în pierderi majore cauzate de furtul de date sau de inserarea unor informaţii false ori denaturate. Analiştii acestui domeniu au sesizat o *contradicţie aparentă* între ***nevoia de comunicaţii şi conectivitate***, pe de o parte şi ***necesitatea asigurării confidenţialităţii, integrităţii şi autenticităţii informaţiilor***, pe de altă parte. Domeniul relativ nou al securităţii informatice caută soluţii tehnice pentru rezolvarea acestei contradicţii. În acest context, asistăm la o nouă tinereţe a domeniului străvechi al criptografiei. Datorită aplicaţiilor ei în securitatea Internet-ului, ***criptografia*** a devenit azi unul dintre cele mai dinamice domenii de cercetare ştiinţifică academică. Acoperită multă vreme sub "voalul" utilizărilor militare şi diplomatice, criptografia este astăzi din ce în ce mai implicată în activităţile noastre informatice de zi cu zi, unii vorbind deja despre existenţa unei criptografii "casnice". De aceea, pe parcursul a câteva numere, vom încerca să explicăm cele mai importante concepte care stau la baza criptografiei moderne şi să vedem care îi sunt principalele aplicaţii în Internet.

Sisteme criptografice simetrice şi cu chei publice

Algoritmii criptografici (cifruri), permit o transformare a datelor numită ***cifrare*** sau ***criptare***. Intrarea unui cifru este ***textul-clar***, iar ieşirea este ***textul-cifrat***. La recepţie, textul-clar se obţine printr-un proces de ***descifrare (decriptare)***, care primeşte la intrare textul-cifrat. Procesele descrise sunt controlate de *chei*, care reprezintă un parametru al transformării.

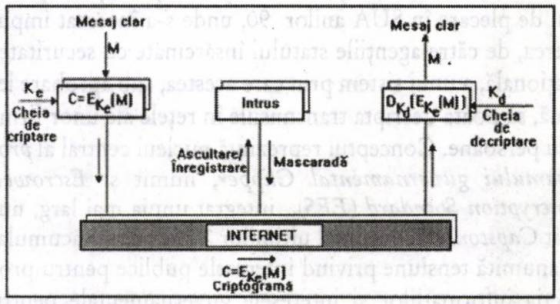


Fig. 1 Criptarea datelor în Internet

Un ***sistem criptografic (criptosistem)***, a cărei ilustrare este făcută în fig. 1, este compus din:

- M-text (mesaj) clar ;
- C-text (mesaj) cifrat sau criptogramă;
- două funcţii inverse $E(\,)$ şi $D(\,)$;
- un algoritm care produce cheile de cifrare K_e şi de descifrare K_d astfel încât:

⇒ **mesajul cifrat *C* este obţinut din mesajul clar *M*, folosind funcţia de cifrare *E* şi cheia de cifrare K_e: $C=E_{K_e}(M)$**

şi

⇒ **mesajul clar este obţinut din criptograma *C*, folosind funcţia de descifrare *D* şi cheia de descifrare K_d: $M=D_{K_d}(C)$**

Există *două tipuri de sisteme criptografice*:

- ***simetrice*** *(cu cheie secretă)* care folosesc aceeaşi cheie, atât la cifrare cât şi la descifrarea mesajelor, adică $K_e=K_d$.
- ***asimetrice*** *(cu cheie publică)* care folosesc chei distincte de cifrare şi descifrare (dar legate una de alta). Una din chei, K_d este ţinută secretă şi este cunoscută doar de proprietarul ei. A doua cheie (perechea ei), K_e, este făcută publică, de unde şi numele de ***criptografie cu cheie publică***.

Ştiinţa care studiază metodele de criptare şi decriptare este ***criptologia***. Ea are în compunere ***criptografia***, cea care se ocupă cu mijloacele de cifrare/descifrare legale şi ***criptoanaliza***, care studiază metode de spargere a cifrurilor, adică de determinare a mesajelor în clar sau a cheilor din criptograme.

Cifrurile simetrice (cu cheie secretă)

Aceste sisteme de criptare folosesc aceeași cheie *K*, atât la cifrarea cât și la descifrarea mesajelor. Cheia este ținută secretă și este folosită în comun de către *emițător* (cel care cifrează mesajul *M*) și de către *receptor* (cel care descifrează criptograma *C)* (fig. 2). Sistemele simetrice sunt bine cunoscute, conduc la performanțe bune și sunt folosite pentru protecția datelor utilizatorilor. Se pot aminti aici criptosisteme simetrice cunoscute, cum ar fi standardul american de cifrare *DES* (*Data Encryption Standard*) sau sistemul *IDEA* (*International Data Encryption Algorithm*). Securitatea criptării simetrice depinde de protecția cheii; administrarea acestora este un factor vital în securitatea datelor și cuprinde următoarele aspecte: *generarea cheilor, distribuția cheilor* și *memorarea cheilor.* Problema fundamentală este cea a găsirii unor modalități de distribuție sigură, periodică, a cheilor criptografice. În serviciile Internet se utilizează tot canalele rețelei, folosind chei de cifrare a cheilor (numite și *chei master*). Sistemele simetrice, datorită cerințelor de distribuție sigură a cheilor criptografice, sunt greoaie în utilizare în comunitățile mari de utilizatori din Internet.

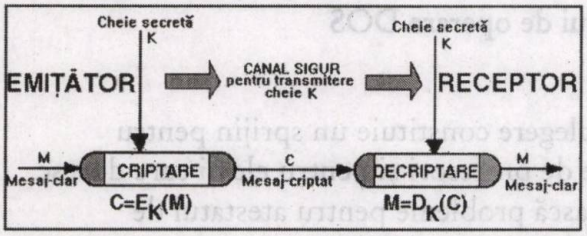


Fig.2 Cifrarea simetrică

Cifrurile asimetrice (cu chei publice)

Sistemele criptografice având chei publice reprezintă un moment crucial în evoluția criptografiei moderne; *Whitfield Diffie* și *Martin Hellman*, de la Universitatea Stanford din California, au pus bazele unui principiu diferit de cel al cifrării simetrice (1976): în locul unei singure chei secrete, criptografia asimetrică folosește două chei diferite, una pentru cifrare, alta pentru descifrare (fig.3). Una dintre chei - ***cheia privată*** (PRIV) - este ținută secretă și este cunoscută doar de proprietarul ei. A doua cheie (perechea ei, dedusă matematic printr-o funcție greu inversabilă din prima) - ***cheie publică*** (PUB) - este făcută publică, de unde și numele de *criptografie cu cheie publică*. Ambele chei sunt de fapt niște fișiere, furnizate de un program capabil să genereze aceste perechi. Dacă cheia publică o puteți da oriunde în lume, pe cea privată trebuie să o păstrați la loc sigur. Pentru a se asigura ***confidențialitatea*** (***secretizarea***) unui mesaj, datele sunt cifrate la emisie cu cheia publică a receptorului. Ele pot fi descifrate doar de către destinatarul autentic, cu cheia privată, pe care o deține doar el. Dacă se dorește ***semnarea digitală*** (***electronică***) a datelor în vederea verificării autenticității, datele sunt cifrate cu cheia privată a emițătorului, iar verificarea poate fi făcută numai cu cheia sa publică (fig.4). Ca urmare, semnătura nu poate fi falsificată, întrucât doar emitentul autentic este în posesia cheii sale private.

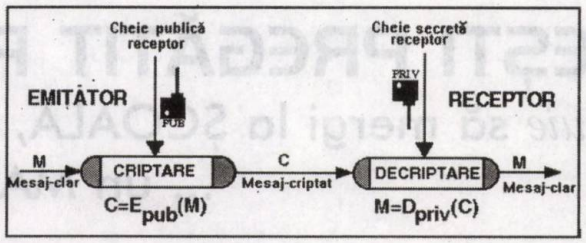


Fig.3 Cifrarea cu chei publice

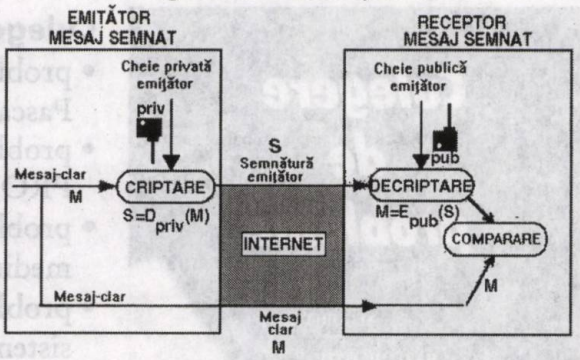


Fig.4 Semnătura digitală (electronică)

O cheie criptografică este de fapt un fișier. Cheia privată "stă" pe calculatorul dumneavoastră pe care nu trebuie să-l părăsească. În ceea ce privește cheia publică, puteți face nenumărate copii pe care să le distribuiți oriunde vreți. Însă dumneavoastră aveți nevoie de amândouă, ele fiind puternic legate una de alta. Folosirea semnăturilor în Netscape se face astfel:

- Un utilizator criptează un mesaj, pe care vi-l adresează, folosind fișierul cheie publică al dumneavoastră, distribuit în mod liber. Apoi trimite mesajul către dumneavoastră. Numai dumneavoastră, folosind fișierul cu cheia privată, puteți decripta și apoi citi mesajul. Emițătorul poate fi sigur că nimeni altcineva nu va putea decripta sau modifica mesajul.
- Dumneavoastră puteți cripta un mesaj cu cheia privată și apoi să-l trimiteți către o altă persoană. Orice persoană care deține cheia dumneavoastră publică (distribuită liber) poate decripta mesajul. În acest caz, persoana poate fi sigură că mesajul provine de la dumneavoastră, singurul care dețineți cheia privată. Este vorba de autentificarea expeditorului prin ***semnătură digitală.***

Algoritmii de criptare cu cheie publică prezintă o cripto-complexitate foarte mare, bazându-se în general pe operații matematice complexe, cu numere întregi foarte mari (sute de cifre zecimale sau mii de biți), ceea ce conferă o tărie deosebită acestor metode de cifrare.

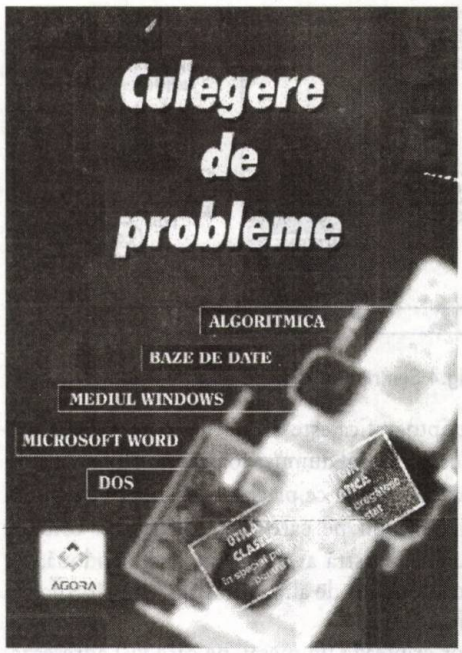
Însă asupra algoritmilor criptografici și a utilizării lor în Internet, în numerele viitoare!

Domnul prof. dr. Victor-Valeriu Patriciu este cadru didactic la Academia Tehnică Militară. Poate fi contactat prin e-mail la adresa: vip@mta.ro

Dacă *încă* nu

EȘTI PREGĂTIT PENTRU MÂINE,

trebuie să mergi la ȘCOALĂ, *trebuie* să ÎNVEȚI, îți *trebuie* ... un MANUAL.



Culegerea conține:

- probleme de algoritmică, majoritatea rezolvate în Pascal
- probleme de gestiune a bazelor de date (FOX-PRO)
- probleme care verifică îndemânarea de a lucra în mediul Windows și cu editorul Microsoft Word
- probleme de verificare a cunoștințelor aferente sistemului de operare DOS

Această culegere constituie un sprijin pentru colectivele de profesori și pentru elevii care doresc să pregătească probleme pentru atestatul de informatică dar poate fi folosită și în scopul pregătirii pe parcursul celor patru ani de liceu.

Culegerea a fost realizată de un număr mare de profesori cu o bogată experiență în predarea informaticii la toate nivelurile. Astfel, culegerea conține o varietate mare de enunțuri de probleme rezolvate, abordând diferite metode și stiluri de programare.

• • •

Dacă vreți, considerați că participați la o întrecere și încercați să rezolvați aceste probleme (suntem convinși că puteți) mai elegant și cu o eficiență în programare mai ridicată.

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422] sau la Computer Press AGORA [Tel: 065-166516; 01-3309281].

INTERFEȚE în Java

Horia Georgescu

Într-un episod anterior, am văzut că este posibil să declarăm clase abstracte în care toate metodele sunt abstracte. Acest lucru poate fi realizat și prin intermediul interfețelor. În plus, ele reprezintă mecanismul propus de Java pentru tratarea problemelor ridicate de moștenirea multiplă.

Noțiunea de interfață

Interfețele reprezintă o modalitate de a declara un tip constând numai din constante și din metode abstracte. Ca sintaxă, o interfață este asemănătoare unei clase, cu deosebirea că în loc de **class** trebuie precizat **interface**, iar metodele nu au corp, acesta fiind înlocuit cu `';'`.

Se poate spune că interfețele reprezintă numai proiecte, pe când clasele sunt o combinație de proiecte și implementări.

Ca și în cazul claselor abstracte, este evident că nu pot fi create obiecte de tipul unei interfețe.

Câmpurile unei interfețe au în mod implicit modificatorii **static** și **final**, deci sunt constante.

Metodele unei interfețe sunt totdeauna publice și au în mod implicit modificatorul **abstract**. De asemenea ele nu pot fi statice.

Orice interfață este gândită pentru a fi ulterior *implementată* de o clasă, în care metodele interfeței să fie redefinite, adică să fie specificate acțiunile ce trebuie întreprinse. Faptul că o clasă `C` implementează o interfață `I` trebuie specificat prin inserarea informației **implements** `I` în antetul clasei.

Exemplu:

```
interface I {
  char c='a'; int i=0;
  void scrie();
}
class C implements I {
  public void scrie() {
    System.out.println(c+" "+i);
  }
}
```

Observații

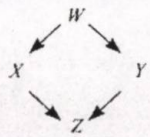
1. O interfață poate extinde oricâte interfețe. În acest mod interfețele permit moștenirea unor contracte (numele și signaturile unor metode) fără moștenirea implementării.
2. Dacă o clasă implementează doar unele din metodele unei interfețe, atunci ea trebuie declarată cu **abstract**.
3. O clasă poate implementa oricâte interfețe, dar poate extinde o singură clasă.

Dacă o clasă extinde o altă clasă și implementează una sau mai multe interfețe, atunci trebuie anunțată întâi extinderea și apoi implementarea, ca în exemplul următor:

```
class C extends B implements I1,I2 { ... }
```

Posibilitatea ca interfețele și clasele să poată extinde oricâte interfețe, reprezintă modalitatea prin care mediul de executare Java rezolvă problema moștenirii multiple.

Dacă s-ar permite ca o clasă să extindă mai multe clase, ar apărea neclarități legate de semnificația numelor. Astfel, dacă ar fi posibilă următoarea structură de clase extinse:



și în clasele `W`, `X` și `Y` ar fi definit (respectiv redefinit) un câmp `c`, atunci pentru un obiect de tipul `Z` semnificația lui `Z.c` ar fi neclară.

De aceea în Java o clasă poate extinde cel mult o clasă. Aceasta nu rezolvă însă automat problema conflictelor de nume.

Suntem acum în măsură să completăm discuția despre *supertipuri și subtipuri.*

Supertipurile unei clase sunt reprezentate de clasa pe care o extinde, interfețele pe care le implementează, precum și supertipurile acestei clase și acestor interfețe. Drept urmare, o referință la un obiect de tipul clasei poate fi folosit oriunde este așteptat un obiect al oricărui supertip al său (vezi și discuția despre *upcasting* din episodul referitor la extinderea claselor).

Mai menționăm că se pot declara variabile având ca tip numele unei interfețe. Unei astfel de variabile îi poate fi asignat un obiect care implementează interfața (are ca tip real o clasă ce implementează interfața) .

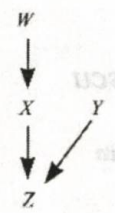
Un exemplu ce ilustrează aceste elemente va fi prezentat în paragraful destinat implementării interfețelor.

Extinderea interfețelor

La fel ca și clasele, și interfețele pot fi extinse.

O interfață `I` poate extinde oricâte interfețe, în acest mod adăugându-se la `I` noi constante și (anunțuri de) metode.

Este permis ca o interfață ce extinde altă interfață să conțină o constantă cu același nume. Fie, de exemplu, următoarea structură de interfețe:



Este posibil ca în fiecare dintre ele să fie declarată o constantă `c`. Dacă o clasă `C` implementează interfața `Z`, atunci o referire la `c` constituie o referire la constanta `c` din `Z`. Putem face însă referire și la constantele din celelalte interfețe prin `X.c`, `Y.c` și `W.c`. Dacă în clasa `C` este redeclarat `c`, este evident că referirea la constanta `c` din `Z` trebuie făcută prin `Z.c`.

Rămânând la structura de interfețe de mai sus, să presupunem acum că în `X` și în `Y` (sau în supertipuri ale lor) apare o metodă cu același nume. Deosebim situațiile:

1) dacă metodele au signaturi diferite, vor fi moștenite ambele metode;
2) dacă metodele au aceeași signatură și același tip pentru valoarea întoarsă, va fi moștenită o singură metodă;
3) dacă metodele au aceeași signatură, dar tipurile valorilor întoarse diferă, atunci moștenirea nu va fi posibilă (eroare de compilare).

De exemplu, este posibil ca în `X` și în `Z` (sau, mai general, într-un drum de interfețe extinse) să anunțăm câte o metodă cu același nume și aceeași signatură. Atunci o clasă `C` care implementează interfața `Z` va implementa ambele metode.

Exemplu:

```
interface W { char c='a'; }
interface X extends W {
  int c=1;
  void met();
}

interface Y { boolean c=true; }
interface Z extends X,Y {
  int c=99;
  void met();
}
class C implements X {
  public void met() {
    System.out.println("++++++");
  }
}
class D implements Z {
  public void met() {
    System.out.println("******");
  }
}
class Interfete1 implements Z {
  public void met() { }
  public static void main (String[] s) {
    C Ob1 = new C(); Ob1.met();
    D Ob2 = new D(); Ob2.met();
    int c=-88;    // met();
    System.out.println(c+" "+W.c+" "+X.c+" "
                       +Y.c+" "+Z.c );
  }
}
```

Programul anterior produce la ieșire:

```
++++++
******
-88 a 1 true 99
```

Să mai observăm că din metoda principală nu se poate apela implementarea lui `met` din `Interfete1`, deoarece o metodă statică poate accesa numai câmpurile și metodele statice ale clasei; pe de altă parte însă, o metodă anunțată într-o interfață nu poate fi redefinită printr-o metodă statică, deci nu putem preciza atributul **`static`** pentru metoda `met` din `Interfete1`.

Implementarea interfețelor

O clasă poate implementa oricâte interfețe, dar cel mult o clasă.

În paragraful de față prezentăm modul în care sunt rezolvate conflictele de nume ce pot apărea la implementarea interfețelor.

Fie `C` o clasă care implementează una sau mai multe interfețe și care, eventual extinde o clasă.

Presupunem mai întâi că în toate aceste tipuri apare un câmp (constantă în cazul interfețelor) cu același nume `c`. Atunci:

- dacă `C` extinde numai o clasă sau implementează numai o interfață, atunci o referire la `c` va semnifica o referire la câmpul `c` al superclasei sau la constanta `c` din interfața implementată;
- în caz contrar, în clasa `C` va trebui să specificăm `X.c` (pentru fiecare interfață `X` implementată) și `super.c` (pentru superclasă).

Trecem acum la cazul metodelor. Moștenirea metodelor interfețelor are loc în modul descris la extinderea interfețelor. Drept urmare, toate metodele moștenite din interfețe trebuie implementate.

Dacă o metodă cu același nume și aceeași signatură apare în superclasă și în interfețele implementate, atunci:

- dacă precizăm în `C` o implementare a metodei, ea va constitui o redefinire a metodei din superclasă; la aceasta din urmă putem face însă apel prin **`super`**;
- dacă în `C` nu este precizată o implementare a metodei, atunci metoda (moștenită) din superclasă constituie implementarea metodei din interfețe, cu condiția ca ea să aibă modificatorul **`public`**.

Exemplu:

Vom considera programul:

```
interface A {
  int x=1;
  void scrie();
}
interface B {
  int x=2;
  void scrie();
}
class C {
  int x=3;
  public void scrie() {
    System.out.println(A.x+" "+B.x+" "+x);
  }
}
class D extends C implements A,B {
  int x=4;
}
class Interfete2 {
  public static void main(String[] w) {
    D Ob = new D(); Ob.scrie();
    A Ob1 = new D(); Ob1.scrie();
    System.out.println(""+Ob1.x);
  }
}
```

Acest program produce la ieșire:

```
1 2 3
1 2 3
1
```

Sunt necesare explicații referitor doar la obiectul `Ob1`. El are tipul declarat `A` și tipul real `D`, ceea ce face ca invocarea metodei `scrie` prin intermediul acestui obiect să se refere la metoda `scrie` din `Interfete2`, iar referința `Ob1.x` să se adreseze constantei `x` din interfața `A`.

Clonarea obiectelor

În multe cazuri suntem puși în situația ca plecând de la un obiect `Ob1` deja construit, să fie nevoie să construim un alt obiect `Ob2` care să-l copieze pe `Ob1`, urmând ca în continuare, cele două obiecte să-și desfășoare independent acțiunile. Cu alte cuvinte, vrem să "creăm" o nouă persoană `Ob2` care să cunoască tot ce cunoaște `Ob1`, iar în continuare cele două persoane să acționeze independent.

Încercarea de a rezolva problema prin asignarea `Ob2=Ob1`, este sortită eșecului. Într-adevăr, să considerăm programul următor:

```
class C {
  int x;
  void inc() {
    x++;
  }
}
class Copie {
  public static void main (String[] s) {
    C Ob1 = new C();
    C Ob2 = Ob1;                    // (*)
    System.out.println(Ob1.x+" "+Ob2.x);
    Ob1.inc(); Ob2.inc(); Ob2.inc();
    System.out.println(Ob1.x+" "+Ob2.x);
  }
}
```

Acest program intenționează să creeze în `Ob2` o copie a obiectului `Ob1`, după care fiecare obiect modifică (prin incrementare) câmpul său `x`. Ținând cont că la crearea unui obiect de tipul `C`, câmpul său `x` este inițializat cu `0`, rezultatele așteptate sunt:

```
0 0
1 2
```

Executarea programului produce însă la ieșire:

```
0 0
3 3
```

Explicația este simplă, cunoscând faptul că un obiect este o referință către adresele câmpurilor și metodelor sale.

Atribuirea `Ob2=Ob1;` realizează doar crearea unui nou nume (un "alias") pentru obiectul `Ob1`.

Nici înlocuirea liniei `(*)` prin

```
C Ob2=new C(); Ob2=Ob1;          (*)
```

nu modifică rezultatele produse la ieșire. De această dată a fost creat efectiv un nou obiect, dar prin atribuirea `Ob2=Ob1;` obiectul `Ob2` constă din aceleași referințe ca și `Ob1`; în particular cele două obiecte vor incrementa aceeași variabilă `x` (ambele folosesc aceeași adresă pentru `x`).

Pentru realizarea clonării obiectelor, Java pune la dispoziție interfața `Cloneable` și metoda `clone` fără parametrii a clasei "rădăcină" `Object`.

Apelarea metodei `clone` de către obiectul curent întoarce un obiect nou, în ale cărui câmpuri sunt copiate câmpurile obiectului curent. Metoda `clone` poate lansa excepția :

`CloneNotSupportedExeption`.

Interfața `Cloneable` din `java.lang` nu are metode; ea identifică o clasă ca fiind sau nu clonabilă prin `clone`. Ea trebuie să fie implementată de clasa în care este invocată `clone`. Modul tipic de lucru este următorul: dacă `C` este clasa ale cărei obiecte urmează a fi clonate, vom specifica faptul că ea implementează interfața `Cloneable` și vom include o metodă care invocă pe `clone`.

Metoda `Object.clone` verifică dacă clasa a cărei instanțiere este obiectul curent, implementează `Cloneable`. În caz negativ, este lansată excepția specificată mai sus. În caz pozitiv, este întors un obiect nou ale cărui câmpuri sunt inițializate ca având aceleași valori ca și câmpurile obiectului curent.

De exemplu, programul care realizează ceea ce intenționase să facă programul precedent are următoarea formă:

```
class C implements Cloneable {
  int x;
  void inc() {
    x++;
  }
  C Clone() {
    try {
      C ObNou = (C) clone();
      return ObNou;
    }
    catch (CloneNotSupportedException e) {
      return null;
    }
  }
}
class Copie1 {
  public static void main (String[] s) {
    C Ob1=new C();
    C Ob2 = Ob1.Clone();
    System.out.println(Ob1.x+" "+Ob2.x);
    Ob1.inc(); Ob2.inc(); Ob2.inc();
    System.out.println(Ob1.x+" "+Ob2.x);
  }
}
```

Vom face următoarele precizări:

- este obligatorie conversia explicită a rezultatului întors de `clone` la tipul dorit;
- am inclus invocarea lui `clone` într-o metodă `Clone` din clasa `C` deoarece aici îi este în mod natural locul (este o prelucrare ce se referă la clasa `C`); pe de altă parte nu se putea invoca `clone` din metoda principală, deoarece nu este statică.

Executarea programului conduce acum la rezultatele dorite:

```
0 0
1 2
```

Continuăm cu un ultim exemplu. De această dată dorim să clonăm un obiect care gestionează o stivă. Dorim ca noul obiect să plece de la același conținut al stivei, dar în continuare lucrează independent, adică stivele gestionate de cele două obiecte să fie distincte.

Stiva este memorată în vectorul `v` și gestionată prin intermediul metodelor `pop` (extragere) și `push` (adăugare în stivă). Constructorul clasei `Stiva` creează vectorul de lungimea transmisă ca parametru și inițializează câmpul `top` cu `-1` (`top` este poziția corespunzătoare vârfului stivei). Metoda `Clone` are aceleași rol ca în programul precedent, iar metoda `scrie` produce la ieșire conținutul stivei.

Pentru moment vom face abstracție de comentariul de sfârșit de linie din metoda `Clone`.

```
class Stiva implements Cloneable {
  private int[] v;
  private int top;
  Stiva Clone() {
    try {
      Stiva ObNou = (Stiva) super.clone();
          // ObNou.v = ( int[] ) v.clone();
      return ObNou;
    }
    catch (CloneNotSupportedException e) {
      return null;
    }
  }
  Stiva(int nrmax) {
    v=new int[nrmax];
    top=-1;
  }
  void push(int val) {
    v[++top]=val;
  }
  int pop() {
    return v[top--];
  }
  void scrie() {
    for (int i=0; i<=top; i++)
      System.out.print(v[i]+" ");
    System.out.println();
  }
}
class Clonare {
  public static void main(String[] s) {
    Stiva S1 = new Stiva(10);
    S1.push(1); S1.push(3);
    S1.push(5); S1.push(7);
    Stiva S2=S1.Clone();
    S2.pop(); S2.pop(); S2.push(13);
    System.out.print("Stiva 1 : ");
    S1.scrie();
    System.out.print("Stiva 2 : ");
    S2.scrie();
  }
}
```

La ieșire va apărea:

```
1 3 13 7
1 3 13
```

care nu este rezultatul dorit. Explicația constă în faptul că la clonare, la copierea câmpurilor în noul obiect `S2`, a fost copiată *adresa* vectorului `v` (tipul tablou este un tip referință), deci se lucrează cu aceeași stivă; câmpul `top` este însă distinct pentru cele două obiecte.

Pentru a lucra cu stive distincte, `v` va trebui clonat explicit. Aceasta se realizează transformând comentariul din `Clone` în instrucțiune efectivă. Cu această modificare, programul va produce acum la ieșire rezultatele dorite:

```
1 3 5 7
1 3 13
```

Prof. dr. Horia Georgescu este cadru didactic la Universitatea București, poate fi contactat prin e-mail la adresa: hg@colegiu.cs.unibuc.ro

Pagini WEB cu HTML 4.0

Mihai Scorțaru

Ce facilități noi ne oferă versiunea HTML 4.0? Cum se lucrează cu frame-uri și cu documente frame-set? Sperăm să învățați ușor...

Noutăți în familia HTML

După cum observați, titlul serialului despre proiectarea paginilor WEB a suferit o mică modificare, fiind înlocuită versiunea HTML pe care o vom studia.

Acest fapt se datorează definitivării unui nou standard HTML, folosit în proiectarea unei noi generații de pagini WEB - cea de-a patra. Versiunea HTML 4.0 a creat "fundația" pentru lucrul cu diverse ***tipuri de media***, dar nu a definit extensiile care trebuie utilizate cu aceste tipuri de media suplimentare. Această lipsă este suplinită de specificația Cascading Style Sheets (CSS), ajunsă deja la versiunea 2, care definește extensiile specifice pentru tipurile de media adiționale.

Probabil unii dintre dumneavoastră nu sunt familiarizați cu termenul de *tip de media*, de aceea vom încerca să dăm o definiție pe înțelesul tuturor. În principiu, un ***tip de media*** este *orice dispozitiv prin intermediul căruia pot fi obținute informațiile conținute într-o pagină WEB*. Observați că nu am folosit termenul de *vizualizare* a paginilor WEB, deoarece tipurile de media nu implică neapărat obținerea informațiilor pe cale vizuală. Pentru a clarifica lucrurile să dăm câteva exemple de tipuri de media: monitorul calculatorului (cel mai des folosit la ora actuală), ecranul televizorului, hârtia (paginile WEB pot fi listate, deși acest lucru nu este recomandabil în cazul în care pagina a fost proiectată fără să se folosească CSS), tipuri de media destinate auzului (boxe active), precum și tipuri de media bazate pe alfabetul Braille, destinate nevăzătorilor.

Datorită marii varietăți a tipurilor de media și a modului în care au fost concepute versiunile HTML anterioare era imposibil ca o pagină WEB să poată fi proiectată pentru mai multe tipuri de media. Să ne concentrăm atenția asupra a două tipuri "fundamentale" de media: *ecranul* și *hârtia*.

Unul dintre motivele pentru tipărirea paginilor WEB care duce la rezultate nesatisfăcătoare este faptul că ecranul calculatorului este un mediu continuu, în timp ce forma tipărită este pe pagini. Datorită continuității reprezentării paginilor pe ecran utilizatorul nu trebuie decât să ruleze conținutul folosind o simplă bară de rulare. Nu se pune aici problema sfârșitului de pagină sau a altor elemente de aspect care determină calitatea unui document tipărit. De exemplu, s-ar putea ca o pagină să conțină un singur rând, ceea ce nu este acceptabil pentru un document care se vrea de calitate.

O altă problemă apare la listarea imaginilor când este posibil ca o imagine să nu încapă în pagina curentă și să fie nevoie să fie listată pe pagina următoare, rămânând astfel un spațiu gol, lucru care nu este acceptabil pentru un document profesional. O soluție propusă a fost "ruperea" imaginii în două, o parte fiind afișată în partea de jos a unei pagini, iar cealaltă în partea de sus a paginii următoare dar, evident, nici această soluție nu a fost agreată. Folosind CSS-2 veți putea defini exact modul în care va arăta pagina WEB tipărită. Acest lucru poate fi realizat prin definirea modului în care imprimanta va prelucra fiecare pagină tipărită, putând fi configurate marginile și sfârșitul fiecărei pagini.

Totuși, specificația CSS-2 nu a fost încă standardizată, dar în următoarele luni, odată cu finalizarea ei, numele acesteia va fi auzit din ce în ce mai des în rândul proiectanților de pagini WEB.

Extensii pentru poziționarea exactă a conținutului unei pagini WEB au fost introduse atât în Internet Explorer 4.0, cât și în Netscape Navigator 4.0, dar căile alese pentru implementarea lor sunt diferite, aceasta ducând la confuzie în rândul proiectanților de pagini WEB.

Versiunea HTML 4.0 introduce o gamă variată de posibilități care pot fi exploatate de proiectanți, dar păstrează compatibilitatea cu versiunile anterioare, deci tot ceea ce s-a spus despre HTML 3.2 este valabil și pentru HTML 4.0.

După cum probabil știți, a apărut deja versiunea Internet Explorer 5.0, iar apariția versiunii Netscape Navigator 5.0 nu va întârzia prea mult (probabil că, în momentul citirii rândurilor de față, această versiune poate fi instalată pe calculatorul dumneavoastră). În scurt timp aceasta va duce la apariția unui standard pentru proiectarea paginilor WEB care va fi denumit HTML 5.0 și, implicit, la apariția unei a cincea generații de pagini WEB, dar până la definitivarea acestui standard, ne vom mulțumi cu prezentarea versiunii HTML 4.0.

Nu se poate spune despre HTML 4.0 că reprezintă o revoluție în lumea paginilor WEB, diferențele față de HTML 3.2 constând, în cea mai mare măsură, din câteva tag-uri noi și câțiva noi parametri pentru tag-urile deja existente. Aceasta înseamnă că un proiectant obișnuit cu vechea versiune va putea trece destul de ușor la noua versiune.

Lucrurile nu sunt la fel de simple în cazul folosirii CSS pentru că această specificație introduce unele concepte noi privind poziționarea conținutului, dar în cele din urmă, tranziția la noul standard HTML va fi destul de ușoară.

Datorită faptului că noile facilități au fost introduse de puțin timp, navigatorii pe Internet care folosesc browsere mai vechi nu vor putea beneficia de ele. Din această cauză, pentru a putea vizualiza pagini WEB proiectate folosind HTML 4.0, trebuie să aveți o versiune a browserului care recunoaște noile elemente introduse. Pentru browserele livrate de firmele Microsoft și Netscape versiunile corespunzătoare sunt: Internet Explorer 4.0 și Netscape Navigator 4.0. În momentul scrierii acestui articol erau deja disponibile pentru download gratuit versiunile Netscape Communicator 4.5 și Internet Explorer 5.0, care recunosc și ele noile tag-uri introduse în HTML 4.0.

Mai mult, după cum probabil știți, noul sistem de operare Windows 98 conține Internet Explorer 4.0. Pentru a încerca să țină pasul cu Microsoft, firma Netscape Communications oferă deja celor interesați codurile sursă pentru viitoarea versiune Netscape Navigator 5.0.

Așadar, nu ar trebui să existe probleme în privința obținerii unui browser care să suporte standardul HTML 4.0. Toți cei care întâmpină dificultăți în acest sens pot contacta redacția pentru informații suplimentare. Dacă, din diferite motive, nu puteți să preluați aceste noi versiuni de pe Internet, vă informăm că majoritatea revistelor de specialitate și nu numai, includ, în cazul în care sunt însoțite de un CD, și cele mai noi versiuni disponibile ale browserelor Internet Explorer și/sau Netscape Navigator.

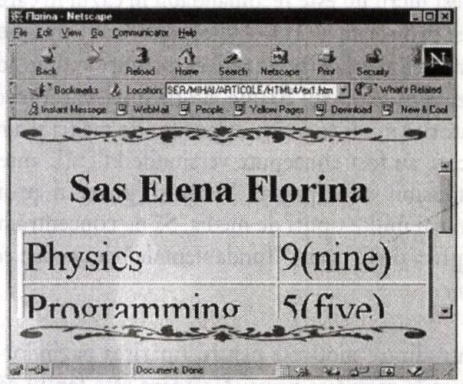
Utilizatorii sistemului de operare Windows 95 ar trebui să știe că, în cazul în care instalează Internet Explorer 4.0 sau o versiune mai nouă, există o opțiune de instalare care, dacă este selectată, duce la diferite modificări ale sistemului (mai ales ale interfeței cu utilizatorul) care s-ar putea să nu fie dorite.

Sperăm că această schimbare a versiunii studiate nu vă va afecta prea mult, mai ales că tot ceea ce a fost prezentat în episoadele anterioare rămâne valabil. Folosirea facilităților oferite de noul standard HTML 4.0 va da un plus de profesionalism paginilor dumneavoastră făcându-le mai interesante, mai atractive și, nu în ultimul rând, reducând timpul necesar încărcării lor.

Acestea fiind zise este timpul să revenim la prezentarea limbajului HTML. Trebuie precizat faptul că marea majoritate a lucrurilor prezentate în paginile următoare sunt valabile și pentru HTML 3.2.

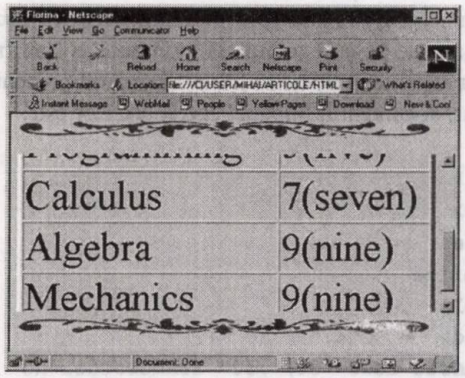
Frame-uri

Una dintre cele mai deranjante limitări ale versiunilor HTML mai vechi era imposibilitatea de a vizualiza mai multe pagini în același timp. Pentru a suplini această lipsă au fost introduse *frame*-urile, fereastra browser-ului fiind împărțită în secțiuni care conțin fiecare, câte un document HTML (o pagină WEB). Fiecare dintre aceste secțiuni poartă denumirea de **frame**. Asemeni tabelelor, *frame*-urile permit aranjarea textului și imaginilor pe linii și coloane, dar spre deosebire de o celulă a unui tabel, un *frame* poate conține link-uri care pot schimba conținutul *frame*-urilor din fereastră. De exemplu, un *frame* ar putea avea rolul unui cuprins, conținutul său neputându-se modifica, dar în funcție de link-ul selectat în acest *frame*, conținutul celorlalte *frame*-uri se modifică. Browserele mai vechi nu suportă această facilitate, *frame*-urile neapărând decât odată cu Netscape Navigator 2.0 și Internet Explorer 3.0. *Frame*-urile reprezintă, în principiu, o modalitate de a prezenta simultan mai multe pagini WEB. Un **frame** se definește ca fiind *o zonă dreptunghiulară a ferestrei browserului, zonă în care este afișată o pagină WEB, în timp ce alte pagini WEB sunt afișate în alte astfel de zone.* La prima vedere figura următoare arată ca o pagină WEB obișnuită dar ea conține trei pagini, fiecare dintre ele fiind afișată într-un *frame* propriu.



Prima deosebire esențială față de o pagină WEB "nor-

mală" este faptul că bara de rulare verticală nu permite rularea conținutului întregii ferestre. Dacă veți încerca să vedeți ce mai conține pagina, veți obține ceva asemănător cu următoarea imagine:



După cum se observă în cele două imagini conținutul din partea de jos și din partea de sus a ferestrei nu se modifică. Aceasta se datorează faptului că din cele trei pagini care sunt afișate în fereastră s-a rulat numai conținutul paginii din mijloc.

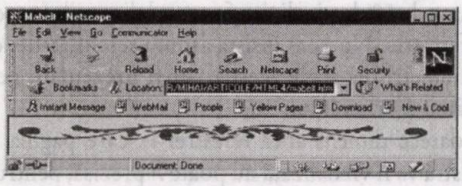
Documente Frameset

Pentru a realiza o imagine de tipul celor de mai sus trebuie creată, mai întâi, fiecare pagină care va fi afișată în fiecare *frame*. Pentru documentul de mai sus au fost create trei fișiere HTML pe care le vom prezenta în continuare.

Fișierul corespunzător *frame*-ului din partea superioară a imaginii este:

```
<html>
<head>
<title>
  Mabelt
</title>
</head>
<body bgcolor = "ffccff">
<div align = "center">
<img src = "mabelt.gif">
</div>
</body>
</html>
```

Pagina WEB corespunzătoare acestui fișier ar arăta astfel:



Pentru partea inferioară a imaginii, fișierul HTML este aproape identic cu cel prezentat mai sus, singura diferență constând în numele fișierului care conține imaginea.

```
<html>
<head>
<title>Mabelb</title>
</head>
<body bgcolor = "ffccff">
<div align = "center">
<img src = "mabelb.gif">
</div>
</body>
</html>
```

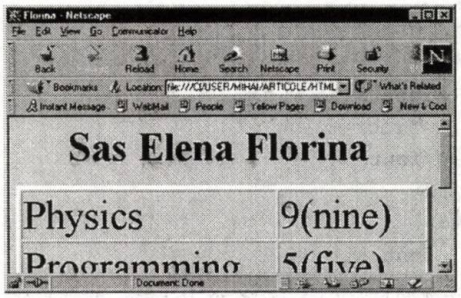
Pagina WEB corespunzătoare este:

Pentru partea din mijloc a imaginii fișierul HTML este următorul:

```
<html>
<head>
<title>
  Florina
</title>
</head>
<body bgcolor = "ffccff">
<div align = "center">
<h1>
<font size = 7>
Sas Elena Florina
</font>
</h1>
<table border = 5 width = 500>
  <tr>
    <td width = 420>
      <font size = 7>
        Physics
      </font>
    </td>
    <td width = 80>
      <font size = 7>
        9(nine)
      </font>
    </td>
  </tr>
  <tr>
    <td>
      <font size = 7>
        Programming
      </font>
    </td>
    <td>
      <font size = 7>
        5(five)
```

```
      </font>
    </td>
  </tr>
  <tr>
    <td>
      <font size = 7>
        Calculus
      </font>
    </td>
    <td>
      <font size = 7>
        7(seven)
      </font>
    </td>
  </tr>
  <tr>
    <td>
      <font size = 7>
        Algebra
      </font>
    </td>
    <td>
      <font size = 7>
        9(nine)
      </font>
    </td>
  </tr>
  <tr>
    <td>
      <font size = 7>
        Mechanics
      </font>
    </td>
    <td>
      <font size = 7>
       9(nine)
      </font>
    </td>
  </tr>
</table>
</div>
</body>
</html>
```

Partea superioară a paginii WEB corespunzătoare acestui fișier arată astfel:



După ce am creat aceste fișiere tot ceea ce avem de făcut este să informăm browserul că dorim ca paginile WEB corespunzătoare să fie afișate în aceeași fereastră. Pentru aceasta va trebui să creăm un document special HTML numit document *frameset*. Un astfel de document nu conține nimic, el având doar rolul de a informa browserul care sunt paginile pe care trebuie să le încarce și cum trebuie acestea aranjate în fereastră. Pentru fereastra prezentată la început, documentul *frameset* corespunzător este:

```
<html>
<head>
<title>
  Florina
</title>
</head>
<frameset rows="50,*,50" frameborder=no
border = 0>
<frame src = "mabelt.htm">
<frame src = "florina.htm">
<frame src = "mabelb.htm">
</frameset>
</html>
```

Deosebirea esențială dintre un document HTML obișnuit și un document *frameset* este înlocuirea tag-ului **<BODY>** cu tag-ul **<FRAMESET>**. Între opening tag-ul și closing tag-ul comenzii **<FRAMESET>** nu poate apărea nici un tag care poate fi conținut între opening tag-ul și closing tag-ul comenzii **<BODY>**. Tag-ul **<FRAMESET>** din exemplul anterior are un atribut **ROWS** care indică faptul că *frame*-urile trebuie așezate unul deasupra celuilalt asemeni liniilor orizontale ale unui tabel. Dacă doriți ca *frame*-urile să fie așezate unul lângă altul trebuie să folosiți în loc de **ROWS** atributul **COLS**. Dimensiunea *frame*-urilor trebuie precizată fie cu valori precise, fie cu procente din dimensiunea ferestrei. Poate fi folosit și simbolul '*' pentru a indica faptul că *frame*-ul respectiv va ocupa tot spațiul rămas în fereastră. Dacă valoarea corespunzătoare mai multor *frame*-uri este '*' atunci spațiul rămas va fi împărțit în mod egal între aceste *frame*-uri.

Comanda <**frameset** rows = "50, *, 50"> indică browserului că fereastra trebuie împărțită în trei regiuni orizontale, așezate una deasupra celeilalte. *Frame*-urile din partea superioară și din partea inferioară vor avea, fiecare, o înălțime de 50 de pixeli, iar *frame*-ul din mijloc va ocupa restul ferestrei. Nu vom insista asupra semnificației celorlalți parametri deoarece aceștia vor fi prezentați, pe larg, spre sfârșitul articolului.

Deoarece dimensiunea ferestrei în care pagina dumneavoastră va fi vizualizată nu poate fi precisă, pentru a indica dimensiunea *frame*-urilor se recomandă în locul valorilor exacte, folosirea procentelor. De exemplu pentru o fereastră care va trebui să conțină două *frame*-uri, dintre care unul trebuie să fie de trei ori mai mare decât celălalt,

trebuie folosită o comandă de genul: <frameset cols= "25%, 75%">. Face excepție de la această regulă cazul în care un *frame* conține o singură imagine care are o dimensiune precizată, caz în care va trebui să precizați dimensiunea *frame*-ului în pixeli. Este cazul exemplului în care imaginile din partea superioară și din partea inferioară a ferestrei au o înălțime de exact 50 de pixeli. În cazul în care doriți să precizați dimensiunile *frame*-ului în pixeli, trebuie să existe cel puțin un *frame* cu dimensiune variabilă (valoarea corespunzătoare dimensiunii trebuie să fie '*') pentru ca documentul să poată fi vizualizat indiferent de dimensiunile ferestrei.

Comanda <FRAME>

Între opening tag-ul și closing tag-ul comenzii **<FRAMESET>** trebuie să apară mai multe tag-uri **<FRAME>** care să indice documentul HTML care va fi afișat în fiecare *frame*. Dacă există mai puține tag-uri **<FRAME>** decât numărul de *frame*-uri definite în tag-ul **<FRAMESET>**, toate *frame*-urile rămase vor rămâne albe. Comanda **<FRAME>** nu are un closing tag.

Fiecare tag **<FRAME>** trebuie să conțină un atribut **SRC**, atribut care indică adresa paginii WEB care trebuie încărcată în *frame*-ul respectiv. În cazul în care doriți ca *frame*-ul respectiv să conțină o singură imagine, acest lucru este posibil precizând adresa fișierului care conține imaginea.

Un *frame* poate conține orice document HTML. Totuși, pentru *frame*-uri de dimensiuni mici se recomandă crearea unor documente special concepute pentru a fi afișate în astfel de *frame*-uri.

Link-uri între frame-uri și ferestre

Cele mai interesante "efecte" pot fi obținute în cazul în care este precizată valoarea atributului **NAME** al tag-ului **<FRAME>**. Dacă un *frame* are un nume atunci orice link poate schimba conținutul acestui *frame* folosind tag-ul **<A>** cu parametrul **TARGET**. Să considerăm documentul *frameset* următor:

```
<html>
<head>
<title>
  Poetry
</title>
</head>
<frameset rows="*,100">
<frame src="delay.htm" name="main">
<frame src = "contents.htm">
</frameset>
</html>
```

Trebuie să creăm acum fișierele **DELAY.HTM** și **CONTENTS.HTM**. Să presupunem că primul fișier este următorul:

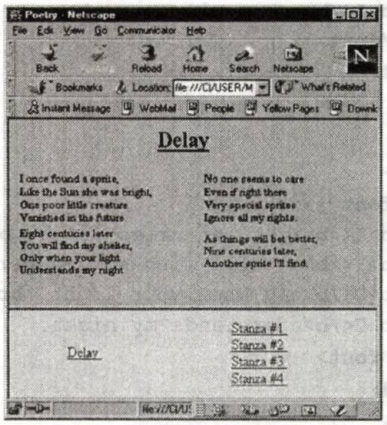
```
<html>
<head>
<title>
  Delay
</title>
</head>
<body bgcolor = "abcdef">
<h2 align="center"> <u>Delay</u> </h2>
<table width = 400 border = 0>
  <tr>
    <td>
      <font size = 2>
      <b>I</b> once found a sprite, <br>
      <b>L</b>ike the Sun she was bright,<br>
      <b>O</b>ne poor little creature <br>
      <b>V</b>anished in the future.
      </font>
    </td>
    <td>
      <font size = 2>
      <b>N</b>o one seems to care <br>
      <b>E</b>ven if right there <br>
      <b>V</b>ery special sprites <br>
      <b>I</b>gnore all my rights.
      </font>
    </td>
  </tr>
  <tr>
    <td>
      <font size = 2>
      <b>E</b>ight centuries later <br>
      <b>Y</b>ou will find my shelter, <br>
      <b>O</b>nly when your light <br>
      <b>U</b>nderstands my night.
      </font>
    </td>
    <td>
      <font size = 2>
      <b>A</b>s things will bet better, <br>
      <b>N</b>ine centuries later, <br>
      <b>A</b>nother sprite I'll find.
      </font>
    </td>
  </tr>
</table>
</body></html>
```

Fișierul **CONTENTS.HTM** este următorul:

```
<html>
<head>
<title>
  Contents
</title>
</head>
<body bgcolor = "abefcd">
```

```
<table width = 400 border = 0>
  <tr>
    <td align = "middle">
      <a href="delay.htm" target="main">
       Delay </a>
    </td>
    <td align = "center">
      <a href = "stanza1.htm" target = "main">
          Stanza #1 </a> <br>
      <a href = "stanza2.htm" target = "main">
          Stanza #2 </a> <br>
      <a href = "stanza3.htm" target = "main">
          Stanza #3 </a> <br>
      <a href = "stanza4.htm" target = "main">
          Stanza #4 </a> <br>
    </td>
  </tr>
</table>
</body>
</html>
```

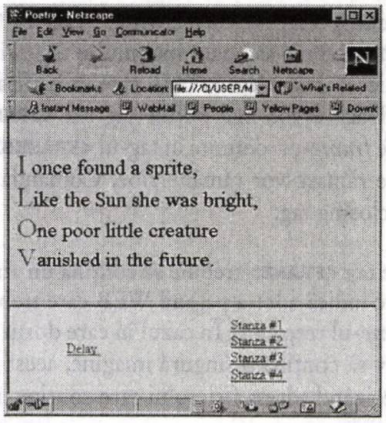
După încărcarea documentului *frameset* pe ecran va apărea următoarea imagine:



Pentru ca link-urile din partea dreaptă să funcționeze, trebuie create fișierele HTML corespunzătoare. Să presupunem că în urma selectării unuia din cele 4 link-uri, va fi afișată strofa corespunzătoare a poeziei. Vom prezenta numai conținutul unuia dintre cele 4 fișiere deoarece celelalte 3 sunt foarte asemănătoare, singura diferență reprezentând-o versurile care trebuie afișate. Fișierul **STANZA1.HTM** este următorul:

```
<html>
<head>
<title>Stanza1</title>
</head>
<basefont size = 5>
<body bgcolor = "fedcba">
<br>
<font size = 6 color = "red">I</font>
once found a sprite,
<br><font size = 6 color = "red">L</font>
ike the Sun she was bright,
<br><font size = 6 color = "red">O</font>
ne poor little creature
<br><font size = 6 color = "red">V</font>
anished in the future.
</body>
</html>
```

Dacă se selectează link-ul care indică spre acest fișier atunci fereastra browserului se va modifica, în locul întregii poezii din partea superioară fiind afișată prima strofă. Efectul poate fi observat în imaginea următoare:



Dacă am selecta acum link-ul *Delay* ar fi afișată din nou întreaga poezie, singura diferență față de imaginea prezentată la început fiind faptul că link-ul spre prima strofă va avea o altă culoare (va indica faptul că pagina corespunzătoare lui a fost deja vizitată).

Există câteva nume predefinite de *frame*-uri care pot fi folosite ca valori pentru parametrul **TARGET** al tag-ului **<A>**. Aceste nume speciale sunt: *_blank*, *_self*, *_top* și *_parent*, ele indicând faptul că pagina corespunzătoare va fi afișată în funcție de nume, în felul următor:
- într-o fereastră nouă fără nume (_blank);
- în *frame*-ul curent (_self);
- în întreaga fereastră a browser-ului (_top);
- în întreaga zonă corespunzătoare *frame*-ului părinte (_parent).

Orice alt nume care începe cu liniuță de subliniere ('_') va fi ignorat.

Margini, borduri și bare de rulare

Pe lângă parametrul **NAME**, tag-ul **<FRAME>** mai poate avea numeroși alți parametri printre care: **MARGINWIDTH**, **MARGINHEIGHT**, **BORDER**, **SCROLLING** și **NORESIZE**.

Parametrul **MARGINWIDTH** indică numărul pixelilor "albi" la stânga și la dreapta conținutului *frame*-ului. Pentru a indica numărul pixelilor albi de deasupra și de sub conținutul *frame*-ului se folosește parametrul **MARGINHEIGHT**.

În mod normal, orice *frame* care are o dimensiune mai mică decât dimensiunea paginii pe care o conține, va avea bare de rulare. Pentru a forța afișarea sau neafișarea barelor de rulare se folosește parametrul **SCROLLING** ale cărui valori pot fi *YES* sau *NO*. În cazul în care acest parametru are valoarea *YES* vor fi afișate bare de rulare verticale și orizontale chiar dacă nu este nevoie de ele. Dacă valoarea este *NO* atunci barele de rulare nu vor fi afișate chiar dacă ar fi nevoie de ele și nu poate fi văzut conținutul întregii pagini conținute în *frame*.

În mod normal cei care vizualizează pagini WEB care conțin *frame*-uri pot modifica dimensiunea acestora. Modificarea poate avea efecte neașteptate privind aspectul paginii. Dacă nu doriți ca dimensiunile *frame*-urilor să poată fi modificate trebuie să folosiți atributul **NORESIZE** al tagului **<FRAME>**.

Frame-uri imbricate

Imbricând mai multe tag-uri **<FRAMESET>** pot fi obținute efecte interesante. De exemplu, putem împărți fereastra browserului în trei *frame*-uri orizontale și apoi, *frame*-ul din mijloc să-l împărțim în alte trei *frame*-uri verticale.

Pentru aceasta să considerăm documentul *frameset* următor:

```
<html>
<head>
<title>
  Frame-uri imbricate
</title>
</head>
<frameset rows = "40, *, 35">
<frame src = "top.htm" scrolling = no>
<frameset cols = "47%, 6%, 47%">
  <frame src = "left.htm" scrolling = no>
  <frame src = "middle.htm" scrolling = no>
  <frame src = "right.htm" scrolling = no>
</frameset>
<frame src = "bottom.htm" scrolling = no>
</frameset>
```

Cele 5 fișiere de care are nevoie acest document *frameset* sunt:

```
<html>
<head>
<title>
  Top
</title>
</head>
<body>
<div align = "center">
<h3> Funny things
</h3>
</div>
</body></html>
```

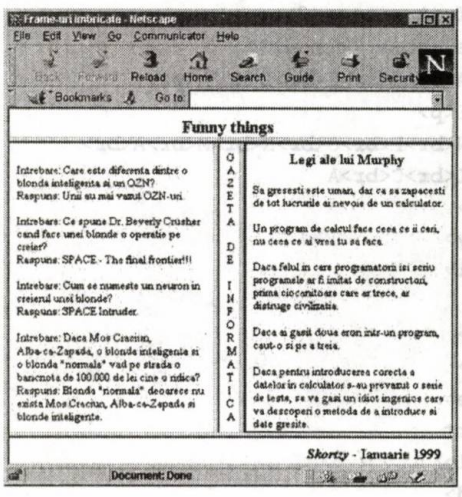
```
<html>
<head>
<title>
  Left
</title>
</head>
<body>
<basefont size=2>
<br>Intrebare: Care este diferenta dintre o
blonda inteligenta și un OZN? <br>
Raspuns: Unii au mai vazut OZN-uri. <p>
Intrebare: Ce spune Dr. Beverly Crusher cand
face unei blonde o operatie pe creier? <br>
Raspuns: SPACE - The final frontier!!! <p>
Intrebare: Cum se numeste un neuron in
creierul unei blonde? <br>
Raspuns: SPACE Intruder. <p>
Intrebare: Daca Mos Craciun, Alba-ca-Zapada,
o blonda inteligenta si o blonda "normala"
vad pe strada o bancnota de 100.000 de lei
cine o ridica? <br>
Raspuns: Blonda "normala", deoarece nu exista
Mos Craciun, Alba-ca-Zapada si blonde
inteligente.
</body>
</html>
```

```
<html>
<head>
<title>
  Middle
</title>
</head>
<body>
<basefont size = 2>
<div align = "center">
G<br>A<br>Z<br>E<br>T<br>A<p>
D<br>E<p>
I<br>N<br>F<br>O<br>R<br>M<br>A<br>
T<br>I<br>C<br>A
</div>
</body>
</html>
```

```
<html>
<head>
<title>
  Right
</title>
</head>
<body>
<basefont size=2>
<div align = "center">
<h4> Legi ale lui Murphy </h4>
</div>
Sa gresesti este uman, dar ca sa zapacesti
```

```
de tot lucrurile ai nevoie de un calculator.
<p>
Un program de calcul face ceea ce ii ceri,
nu ceea ce ai vrea tu sa faca. <p>
Daca felul in care programatorii isi scriu
programele ar fi imitat de constructori,
prima ciocanitoare care ar trece, ar
distruge civilizatia. <p>
Daca ai gasit doua erori intr-un program,
caut-o şi pe a treia. <p>
Daca pentru introducerea corecta a datelor
in calculator s-au prevazut o serie de
teste, se va gasi un idiot ingenios care va
descoperi o metoda de a introduce si date
gresite. <p>
</body>
/html>

<html>
<head>
<title>
  Bottom
</title>
</head>
<body>
<div align = "right">
<b> <i> Skortzy </i>
- Ianuarie 1999
</b>
</div>
</body>
</html>
```

Fereastra browserului va arăta astfel:



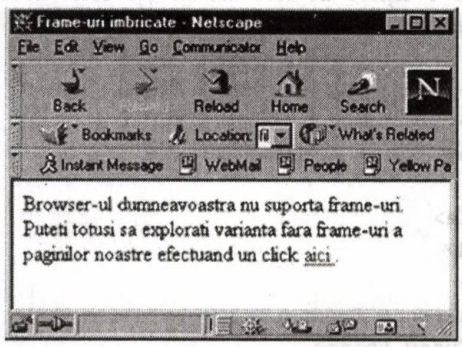
Proiectarea paginilor WEB pentru browserele care nu suportă frame-uri

Există browsere mai vechi care nu suportă *frame*-uri. Pentru ca cei care folosesc astfel de browsere să poată obține informațiile conținute în paginile dumneavoastră, va trebui să creați o versiune fără *frame*-uri. Pentru a da această posibilitate celor care au browsere mai vechi, s-a introdus tag-ul **<NOFRAMES>**. Informația inclusă între opening tag-ul și closing tag-ul acestei comenzi este ignorată de browserele noi, dar browserele vechi nu vor recunoaște această comandă și o vor ignora, executând comenzile care ar fi fost ignorate de browserele noi.

Browserele vechi vor ignora și tag-urile **<FRAMESET>** și **<FRAME>** și vor efectua numai comenzile rămase. Pentru ca aceste browsere să poată executa comenzile necesare trebuie ca tag-ul **<NOFRAMES>** să conțină un tag **<BODY>**. Iată cum se va rezolva această problemă pentru ultimul exemplu prezentat:

```
<html>
<head>
<title>
  Frame-uri imbricate
</title>
</head>
<frameset rows = "40, *, 35">
<frame src = "top.htm" scrolling = no>
  <frameset cols = "47%, 6%, 47%">
  <frame src = "left.htm" scrolling = no>
  <frame src = "middle.htm" scrolling = no>
  <frame src = "right.htm" scrolling = no>
  </frameset>
<frame src = "bottom.htm" scrolling = no>
</frameset>
<noframes>
<body>
Browser-ul dumneavoastra nu suporta frame-
uri. <br>
Puteti totusi sa explorati varianta fara
frame-uri a paginilor noastre efectuand un
click
<a href = "noframe.htm"> aici </a>.
</body>
</noframes>
```

Un browser mai vechi va afișa următoarea imagine:



Mihai Scorțaru este student la Universitatea Tehnică din Cluj-Napoca și poate fi contactat prin e-mail la adresa: scort@licj.soroscj.ro

Concursul INFO-STAR '98

"MOȘ NICOLAE" la Clubul Copiilor

Sorin Groza

Clubul Copiilor din Aiud a organizat ediția a IV-a a concursului județean de informatică INFO-STAR. Și în acest an "piticii" s-au întrecut în rezolvarea problemelor de informatică, probleme propuse de către domnul profesor Ovidiu Domșa de la Colegiul "Horea, Cloșca și Crișan" din Alba Iulia. Vă prezentăm în continuare problemele din concurs, împreună cu rezolvările propuse de către copiii care au obținut un punctaj maxim.

Cadouri de Moș Nicolae

Ana, Emil și Nicu găsesc în ghetele din geam cadouri de la Moș Nicolae. Fiecare dintre ei descoperă printre cadouri câte un loz în plic câștigător a unei sume de bani. Copiii doresc să cumpere de la un prieten o rachetă care costă s *lei. Suma trebuie plătită exact, folosind unul două sau trei dintre lozurile copiilor. Ajutați-i să rezolve problema și afișați pe ecran numele copiilor din banii cărora se poate cumpăra racheta.*

Dacă nu există soluție se va afișa mesajul: 'Copiii nu cumpara racheta!'.

Date de intrare:

500 - câștigul Anei
1000 - câștigul lui Nicu
5000 - câștigul lui Emil
6000 - prețul rachetei

Date de ieșire:

Nicu Emil

Rezolvare

Problema constă în a verifica toate posibilitățile de formare a prețului rachetei, folosind sumele celor trei copii. Se folosește o variabilă "semafor" c, care primește valoarea 1 atunci când se găsește o soluție a problemei și în acest caz se tipărește cel puțin o soluție. Dacă în final această variabilă are valoarea 0 (cea inițială) înseamnă că nu s-a găsit nici o soluție și în acest caz se va tipări mesajul cerut.

Vom prezenta rezolvări implementate în limbajele Basic și Pascal.

Listing: MOS.BAS

```
INPUT "Castigul Anei este"; q
INPUT "Castigul lui Nicu este"; w
INPUT "Castigul lui Emil este"; e
INPUT "Costul rachetei"; s
c=0
IF q=s THEN PRINT "Ana": c=1
IF w=s THEN PRINT "Nicu": c=1
IF e=s THEN PRINT "Emil": c=1
IF q+w=s THEN PRINT "Ana Nicu": c=1
IF q+e=s THEN PRINT "Ana Emil": c=1
IF w+e=s THEN PRINT "Nicu Emil": c=1
IF q+w+e=s THEN PRINT "Ana Nicu Emil": c=1
IF c=0 THEN PRINT "Copiii nu cumpara racheta"
```

Observație:

În varianta implementată în Pascal în locul variabilei c (având două valori posibile) am folosit variabila booleană se_poate.

Listing: MOS.PAS

```
Program Mos_Nicolae;
var q,w,e,s:Integer;
    se_poate:Boolean;

Begin
  Writeln('Castigul Anei este ');
  Readln(q);
  Writeln('Castigul lui Nicu este ');
  Readln(w);
  Writeln('Castigul lui Emil este ');
  Readln(e);
```

```
  Writeln('Costul rachetei ');
  Readln(s);
  se_poate:=false;
  if q=s then
  begin Writeln('Ana'); se_poate:=true end;
  if w=s then
  begin Writeln('Nicu'); se_poate:=true end;
  if e=s then
  begin Writeln('Emil'); se_poate:=true end;
  if q+w=s then
  begin
    Writeln('Ana Nicu'); se_poate:=true
  end;
  if q+e=s then
  begin
    Writeln('Ana Emil'); se_poate:=true
  end;
  if w+e=s then
  begin
    Writeln('Nicu Emil'); se_poate:=true
  end;
  if q+w+e=s then
  begin
    Writeln('Ana Nicu Emil'); se_poate:=true
  end;
  if not se_poate
  then Writeln('Copiii nu cumpara racheta')
End.
```

Cura de slăbire

La un tratament de cură de slăbire de k *zile se înscriu* n *persoane, fiecare dintre ele având o anumită greutate. Greutatea ideală se consideră a fi mai mare sau egală cu* a *și mai mică sau egală cu* b. *Persoanele urmează unul din următoarele tratamente:*

– cei sub greutatea ideală se îngrașă 1 kg pe zi;
– cei peste greutatea ideală slăbesc 1 kg pe zi;
– cei cu greutatea ideală își păstrează greutatea.

Problema cere să se afle:

a) numărul persoanelor cu greutate ideală la începutul tratamentului;
b) numărul persoanelor cu greutate ideală după cele k *zile de tratament.*

Date de intrare:

```
Cate persoane participa la cura? 10
Cate zile dureaza cura? 7
Intervalul de greutate ideala: 40 50
Greutatile persoanelor:
38 41 48 50 52 54 58 60 42 32
```

Date de ieșire:

```
a) 4
b) 7
```

Rezolvare:

Pentru rezolvarea primului punct se parcurge șirul în care sunt păstrate greutățile și se numără câte elemente sunt cuprinse în intervalul [a,b]. În continuare se aplică tratamentul enunțat pentru cele k zile, zi cu zi. Se parcurge de k ori șirul greutăților și se fac modificările corespunzătoare pentru fiecare parcurgere, adică după fiecare zi. Se poate întâmpla ca după un anumit număr de zile o persoană să ajungă la greutatea ideală și astfel tratamentul ei se încheie.

După cele k parcurgeri se numără din nou câte elemente aparțin intervalului [a,b], adică numărul persoanelor cu greutate ideală la sfârșitul tratamentului.

Listing: CURA.BAS

```
INPUT "Cate persoane participa la cura? "; n
DIM g(n)
FOR i=1 TO n
PRINT "greutatea persoanei"; i
INPUT g(i)
NEXT i
INPUT "Cate zile dureaza cura? "; k
INPUT "Intervalul de greutate ideala: "; a, b
nri=0: nrf=0
FOR i=1 TO n
  IF g(i)>=a AND g(i)<=b THEN nri=nri+1
NEXT i
PRINT "a)"; nri
FOR zile=1 TO k
  FOR i=1 TO n
    IF g(i)<a THEN g(i)=g(i)+1
    IF g(i)>b THEN g(i)=g(i)-1
  NEXT i
NEXT zile
FOR i=1 TO n
  IF g(i)>=a AND g(i)<=b THEN nrf=nrf+1
NEXT i
PRINT "b)"; nrf
```

Listing: CURA.PAS

```
Program Tratament;
var n,i,k,a,b,nri,nrf,zile:Integer;
    g:array[1..100] of Integer;

Begin
  Writeln('Cate persoane participa la cura? ');
  Readln(n);
  Writeln('Greutatile persoanelor: ');
  for i:=1 to n do Read(g[i]); Readln;
  Writeln('Cate zile dureaza cura? ');
  Readln(k);
  Writeln('Intervalul de greutate ideala: ');
  Readln(a,b);
  for i:=1 to n do
    if (g[i]>=a) and (g[i]<=b)
    then Inc(nri);
  Writeln('a) ',nri);
```

```
  for zile:=1 to k do
    for i:=1 to n do
    begin
      if g[i]<a then Inc(g[i]);
      if g[i]>b then Dec(g[i])
    end;
  for i:=1 to n do
    if (g[i]>=a) and (g[i]<=b) then Inc(nrf);
  Writeln('b) ',nrf)
End.
```

Jocul eliminării

La un joc participă n *copii așezați în cerc. Se stabilește o persoană* p *de la care va începe jocul și un număr întreg* m. *De la persoana aleasă, numărând în sensul acelor de ceasornic, este eliminată a* m*-a persoană. Cercul se închide din nou și se continuă numărătoarea de la următoarea persoană de după cea eliminată. Repetând acest procedeu sunt eliminate toate persoanele, ultima ieșită din joc fiind declarată câștigătoare.*

a) afișați persoanele în ordinea în care sunt eliminate;

b) stabiliți numele persoanei de la care ar trebui să înceapă numărătoarea, pentru a fi învingătoare persoana k.

Date de intrare:

```
n=5, p=2, m=7, k=3
```

Date de ieșire:

```
a) 3 1 2 4 5
b) Incepe 5
```

Rezolvare

Pentru afișarea persoanelor în ordinea eliminării, se va folosi un șir de lungime n cu elementele a[i]=i. Marcarea unei persoane eliminate i înseamnă că a[i] devine 0. Continuând numărarea, elementele egale cu 0 nu vor fi luate în considerare. Pentru a simula mișcarea circulară, indicele care identifică pozițiile persoanelor va deveni 1 în momentul în care se ajunge la sfârșitul șirului.

Pentru a afla persoana de la care ar trebui să se înceapă jocul se va repeta rezolvarea punctului anterior, variind persoana de la care începe jocul până când persoana introdusă de la tastatură este învingătoare.

Pentru un număr mare de persoane, timpul de răspuns crește. De aceea, pentru punctul b) propunem un algoritm mai simplu: la punctul a) se determină diferența dintre poziția persoanei de la care se începe jocul și poziția persoanei care câștigă jocul. Pentru a afla numele persoanei de la care trebuie să înceapă jocul se va scădea din poziția persoanei care trebuie să câștige, diferența găsită la punctul a).

Listing: JOC.PAS

```
Program Joc;
var n,m,i,j,k,p,nr_iesiti:Byte;
    copil:array[1..100] of Byte;
Begin
  Write('Introduceti numarul de persoane: ');
  Readln(n);
  Write('Introduceti persoana aleasa: ');
  Readln(p);
  Write('Introduceti numarul m: ');
  Readln(m);
  Write('Introduceti k: ');
  Readln(k);
  for i:=1 to n do copil[i]:=i;
  Writeln('Se elimina in ordine: ');
  nr_iesiti:=0; i:=p-1;
  while nr_iesiti<n do
  begin
    j:=0;
    while j<m do
    begin
      if i<n then i:=i+1
             else i:=1;
      if copil[i]<>0 then j:=j+1
    end;
    copil[i]:=0; Write(i);
    nr_iesiti:=nr_iesiti+1
  end
End.
```

În încheiere vă propunem spre rezolvare următoarea problemă:

Centrul de informatică

La Aiud s-a inaugurat noul centru de informatică. Un grup format din n *copii doresc să se înscrie la cursurile organizate la acest centru. Știind că grupurile de cursanți sunt constituite numai din prieteni, cunoscând perechile de prieteni și știind că este valabilă afirmația "Prietenul prietenului meu îmi este prieten și mie", aflați:*

a) numele copiilor care nu au nici un prieten;

b) numele copiilor care au cei mai mulți prieteni direcți (care formează o pereche de prieteni);

c) numele prietenilor unui copil x *citit de la intrare.*

Date de intrare:

```
n=6
1 2
2 3
5 6
x=3
```

Date de ieșire:

```
a) 4
b) 2 cu 2 prieteni directi
c) 1 2
```

Domnul profesor Sorin Groza este cadru didactic la Clubul Copiilor din Aiud și poate fi contactat prin intermediul redacției.

Vă place GInfo?

Nume: _____ Prenume: _____ Vârsta: _____

Adresa: _____

Localitate: _____ Cod poştal: _____ Judeţ: _____ Telefon: _____

Ocupaţia: elev(ă) informatică ☐ elev(ă) ☐ student(ă) ☐ cadru didactic ☐ altă ocupaţie ☐ (precizaţi: _____)

Şcoala (firma, organizaţia etc.): _____

Dotări

Folosiţi un calculator acasă? Da ☐ Nu ☐

Câte calculatoare sunt instalate în şcoală: (aprox.) _____

Aveţi acces la e-mail: Da ☐ Nu ☐

Aveţi acces la WWW: Da ☐ Nu ☐

Cum apreciaţi dotarea cu tehnică de calcul a şcolii: _____

Aprecieri generale

Acordaţi note de la 1 la 10 pe următoarele criterii:

Structura tematică: ___

Calitatea articolelor: ___

Prezentare grafică: ___

Calitate tipografică: ___

Consideraţi că preţul este:

prea mare ☐ accesibil ☐ prea mic ☐

De cât timp cunoaşteţi Gazeta de Informatică (ani)? _____

Cum aţi procurat acest număr:

chioşc ☐ abonament ☐ cadou ☐ altfel ☐

În acest număr

Apreciaţi cu note de la 1 la 10 articolele acestui număr:

Numere prime şi divizibilitate ___

Branch&Bound şi backtracking ___

Criptografia în Internet ___

Microprocesoare ___

Programarea plăcii de sunet ___

Interfeţe în Java ___

Pagini Web cu HTML 4.0 ___

Soft creat de elevi ___

"Moş Nicolae" la Clubul Copiilor ___

Articolul care v-a plăcut cel mai mult: _____

De ce? _____

Articolul care v-a plăcut cel mai puţin: _____

De ce? _____

Consideraţi că ponderea problemelor rezolvate este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că ponderea problemelor propuse este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că ponderea subiectelor teoretice este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că nivelul de accesibilitate al articolelor este

prea coborât ☐ potrivit ☐ prea ridicat ☐

În viitor

În următoarele numere aşteptaţi:

mai multe articole de programare? Da ☐ Nu ☐

mai multe articole teoretice? Da ☐ Nu ☐

mai multe articole despre Internet? Da ☐ Nu ☐

mai multe prezentări de programe? Da ☐ Nu ☐

prezentări de jocuri? Da ☐ Nu ☐

articole despre hardware? Da ☐ Nu ☐

V-ar plăcea să citiţi:

pagini redactate de elevii unor licee? Da ☐ Nu ☐

lucrări premiate, realizate de elevi? Da ☐ Nu ☐

actualităţi şcolare? Da ☐ Nu ☐

prezentarea unor situri Web? Da ☐ Nu ☐

tips & tricks? Da ☐ Nu ☐

Comentarii:

_____

_____

_____

_____

_____

_____

Dacă spaţiul nu vă este suficient, folosiţi coli albe suplimentare

artea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Car

COMPUTER LIBRIS AGORA

COMPUTER LIBRIS AGORA Cluj-Napoca, Str.Universității Nr.8/8, 3400, Tel. 064-192422
COMPUTER PRESS AGORA Tg.Mureș Str.Tudor Vladimirescu Nr. 63/1 (CP 172/1), Tel. 065-166516, Fax 065-166290
Sucursala București: Str.Poterași, Nr.35/2 (CP 94/49), Tel: 01-3309281, Fax: 01-3309285

Un sfat prinde bine...

PC MAGAZINE ROMÂNIA

...oricând!

Dacă vrei să te orientezi printre miile de produse și tehnologii informatice de ultimă oră

ABONEAZĂ-TE!

Câștigi **30 000** lei și vei primi cadou și CD-urile **WINSTONE '99** și **WINBENCH '99**

COMPUTER PRESS AGORA, Str.Tudor Vladimirescu Nr.63/1, 4300 Tg.Mureș, Tel.065-166516, Fax 065-166290

gazeta de informatica

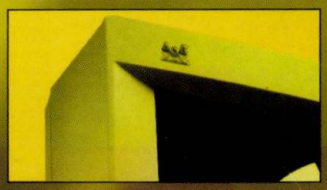
Revista viitorilor specialiști în tehnologia informației

Martie 1999
Vol. 9/3

Lei 10.000

din sumarul numărului:

Calculatoare pe bază de ADN



Structura unei baze de date

TEHNO

Mira
limbaj de programare pentru grafică

SERIAL

Pagini WEB ... 4.0
Fire de executare (Java)

INTERNET

Criptarea simetrică

ACTUAL

Efectele negative ale calculatoarelor

Plus

HTML
VBScrip

Applet-uri Java

BAC

COMPUTER PRESS AGORA

PC & BYTE REPORT

TEHNOLOGIA INFORMAȚIEI

ECHIPAMENTE • SISTEME • APLICAȚII • REȚELE • INTERNET

*Abonează-te și plătești doar 120.000 lei (cu 60.000 lei mai puțin decât prețul de vânzare), evitând și ulterioarele majorări de preț

COMPUTER PRESS AGORA • CP.230 OP.1 TG.MUREȘ, Tel. 065-166516 Fax 065-166290

Vol. 9, Nr. 3 1999

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Editor executiv:
Mircea Sârbu (Târgu-Mureș)
Au mai colaborat:
Gheorghe Păun (București)
Victor-Valeriu Patriciu (București)
Marin Vlada (București)
Ioan Cozac (Tg. Mureș)
Doina Rancea (Cluj)
Mihai Oltean (Cluj)
Mihai Scorțaru (Cluj)

Tehnoredactare:
Dan Crețu, Călin Borz
Coperta și grafica:
Mircea Drăgoi

Adresa pentru corespondență:
C.P. 172-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: (065) 166.516
Fax: (065) 166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *S.C. Infopress srl*,
Odorheiu Secuiesc,
tel: 066.216.483
Editor: Computer Press Agora srl
str. T. Vladimirescu nr. 63/1
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director editorial:
Mircea Sârbu
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (mszabo@agora.ro)
Denis Mărioara (mdenis@agora.ro)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: (01) 330.92.81

Σ sumar

probleme.. PAG. 5

propuse pentru REZOLVARE

Faza regională a concursului Nemes Tihamér

actual.. PAG. 8

Efectele negative ale calculatoarelor asupra noastră
Mihai OLTEAN

bac.. PAG. 10

Bacalaureat 1999
Doina RANCEA

Rezolvarea subiectului publicat în GInfo 2 și... un subiect nou

focus.. PAG. 17

Calculatoare pe bază de ADN
Gheorghe PĂUN

tehno.. PAG. 21

Arhitectura unei baze de date
Ioan COZAC

MIRA - limbaj de programare pentru grafică
Marin VLADA

internet.. PAG. 30

Criptarea simetrică
Victor-Valeriu PATRICIU

Secretele lumii și protecția datelor

serial.. PAG. 36

Fire de executare
Horia GEORGESCU

Pagini WEB cu HTML 4.0
Mihai SCORȚARU

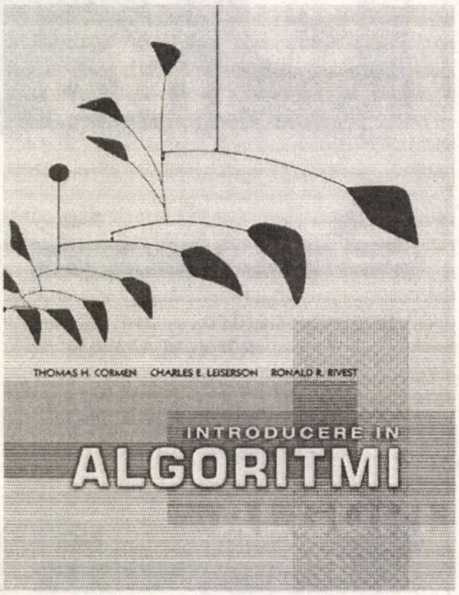
begin.. PAG. 54

Probleme pentru începători

COMPUTER LIBRIS AGORA

În curând o nouă apariție editorială

La începutul lunii mai va apare în sfârșit și în limba română celebra carte de ALGORITMI scrisă de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, publicată de MIT Press, SUA. Este o carte care de la prima ei apariție a fost retipărită, numai în Statele Unite, de 21 ori.



Această carte oferă o introducere exhaustivă a studiului modern al algoritmilor. Analiza și proiectarea algoritmilor tratați este accesibilă și asigură condițiile unui studiu, pe de o parte fundamental, pe de altă parte profund.

Fiecare capitol prezintă cel puțin un algoritm sau o tehnică de proiectare, un domeniu de aplicații, respectiv un subiect înrudit. Algoritmii sunt descriși în limba română și în pseudocod. Cartea conține peste 260 de figuri care ilustrează modul de funcționare al algoritmilor. Deoarece accentul este pus pe eficiență ca și criteriu de proiectare, include analize atente ale timpului de executare a tuturor algoritmilor.

Cartea originală conține 1000 pagini și este structurată în șapte părți, subdivizate în 37 capitole, cu o bibliografie de 205 titluri și o listă bibliografică de 30 pagini.

Partea I, "Fundamente matematice", este un compendiu al metodelor și instrumentelor care vor fi utilizate în analiza algoritmilor. Este organizată esențial pentru referință, ca un fel de tutorial.

Partea a II-a, "Sortare și statistici de ordine", tratează problematica tehnicilor de sortare. Sunt studiate metodele *heapsort, quicksort și tehnici de sortare în timp liniar, precum și statisticile de ordine.*

Partea a III-a, "Structuri de date", prezintă unele tehnici de bază pentru reprezentarea și manipularea mulțimilor dinamice finite. Sunt discutate *structuri de date elementare, tabele de dispersare, arbori de căutare binară, arbori roșu-negru, extinderea structurilor de date.*

Partea a IV-a, "Tehnici avansate de proiectare și implementare", acoperă trei tehnici importante pentru proiectarea și analiza algoritmilor eficienți: *programarea dinamică, algoritmi greedy și analiza amortizată.* Aceste tehnici sunt ceva mai sofisticate decat altele, dar sunt esențiale pentru faptul că permit abordarea efectivă a multor probleme computaționale.

Partea a V-a, "Structuri de date avansate", revine la examinarea structurilor de date care suportă operații pe mulțimi dinamice, dar la un nivel mai avansat decat cel din Partea a III-a. Sunt discutați *B-arbori, ansambluri binomiale, ansambluri Fibonacci, precum și structuri de date pentru mulțimi disjuncte.*

Partea a VI-a, "Algoritmi de grafuri", tratează unele dintre cele mai importante probleme computaționale definite pe grafuri. Sunt trecuți în revistă algoritmi elementari pe grafuri, arbori de acoperire minimă, problematica celor mai scurte drumuri și algoritmi pentru determinarea fluxului maxim.

Partea a VII-a, "Subiecte diverse", conține o selecție de subiecte care extinde și completează materialul din restul cărții. Unele capitole introduc modele de calul noi precum circuite combinatoriale sau computere paralele. Altele acoperă domenii mai specializate precum geometria computațională sau teoria numerelor. Ultimele două capitole discută despre unele dintre limitările cunoscute ale proiectării algoritmilor eficienți și introduc tehnici pentru studierea acestor limitări.

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422]

REFORMA învățământului

De zece ani așteptăm să se întâmple ceva demn și adecvat sfârșitului de secol în care trăim și să putem privi încrezători spre mileniul care vine. O țară întreagă se zbate în căutarea soluțiilor de a trăi mai bine, dar constată că trebuie să se mulțumească cu jumătăți de măsură sau cu un minim îngrijorător.

Nu, nu voi amesteca politică în acest editorial, dar trebuie să încep cu o sintagmă care se pare că nu este cunoscută de către cei de la cârma țării: dacă vrem să asigurăm viitorul unei țări, trebuie găsită calea de a investi la propriu și la figurat în învățământ.

Nu vreau să înșir țări care într-un moment similar celui din '89 din România, au reușit să revigoreze întreaga economie din țara lor, investind, în primul rând în viitorul tineretului, deci în învățământ.

Reforma promisă întârzie, iar timpul nu iartă. Aș adăuga repede: *felicitări Domnului Ministru* ***Andrei Marga****, care, luptându-se cu "mentalități" învechite, cu inerția specifică a unor instituții, a reușit să lanseze în sfârșit proiectele "schimbării"*. Va fi bine? Va fi mai rău? Sunt întrebări la care vom primi răspunsul în anii care vin. Un lucru este sigur: *va fi altfel*!

Informatica în noul plan de învățământ

În România există o tradiție în ceea ce privește predarea informaticii în învățământul preuniversitar. Din 1972 există licee în care în 8 ore pe săptămână s-a predat informatica. Pe parcursul celor 26 de ani care au trecut, numărul celor care au urmat liceul la secția "informatică" a crescut continuu. Era și firesc să se întâmple așa.

Calculatorul, "unealtă modernă de lucru", *a atras* și *atrage* în continuare prin faptul că promite un loc de muncă elegant, curat, unde se pot obține rezultate repede; eficiența este asigurată însăși prin faptul că nu se lucrează cu "ciocanul".

Acum, în momentul în care pe masa de lucru există definitivat deja nu doar planul de învățământ, ci și noile programe școlare pentru clasele de gimnaziu și pentru clasa a IX-a, ne putem întreba, de ce a fost aleasă această cale? Liceele teoretice vor fi de patru feluri, după cum se știe din clipa popularizării noului plan de învățământ oferind posibilitatea de a alege printre altele specializarea matematică-informatică.

La prima vedere aici se ascunde un mare câștig: informatica a devenit știință și este pusă în pereche cu matematica. Înseamnă că învățământul devine mai pragmatic, se orientează în funcție de cererea pieții.

Totuși, mulți directori de școală, profesori de informatică, dar și elevi se îngrijorează în momentul în care văd că în acest plan nu mai sunt 8 ore de informatică pe săptămână, ci doar 3 în clasa a IX-a, 4 în clasa a X-a și a XI-a, respectiv 5 în a XII-a. Celor care au cerut mai multe ore, li s-a explicat că mai multe ore la o singură disciplină sunt posibile doar în filiera tehnică, iar acolo sunt o serie de alte criterii care îngrădesc dezvoltarea unui învățământ ambițios.

Las la o parte această nemulțumire, deoarece ea, evident, se naște în momentul în care cineva își pune întrebare: "Cum se va putea preda în 3 ore ceea ce s-a predat până acum în 8 ore?" ***Nu va trebui să se predea ceea ce s-a predat!*** Predarea și învățarea disciplinei va urma *alte obiective*, pe un *alt conținut*, într-un *alt ritm*.

Câștigul

Cel mai important mi se pare, în schimb, apariția, în planul de învățământ, la alte profiluri, a disciplinei *Tehnologia Informației*.

În anul 2000 învățarea informaticii nu trebuie să rămână un privilegiu al unora care întâmplător (sau din diferite motive) s-au înscris într-o clasă de informatică.

Tot mai puține domenii de activitate se pot "lăuda" că nu au nevoie de calculator, pe de altă parte sunt foarte puține - în care să fie nevoie de cunoștințe de "algoritmică", în schimb este nevoie de cunoașterea diferitelor pachete de soft, a utilitarelor.

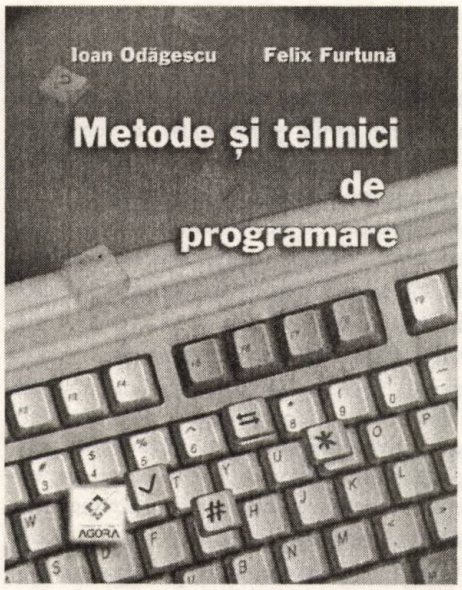
Nicăieri în lume nu există încă noțiunea de "informatică pură", dar, probabil azi-mâine vom ajunge în situația în care cunoștințele de ordin științific va trebui să se separe de cele ale utilizatorilor, altfel *X* miliarde de persoane de pe Pământ se vor putea considera informaticieni. Nu cred că acest lucru va constitui o problemă, dar mă întreb cum se vor numi cei care vor dezvolta la nivel științific, mai departe, informatica?

Clara Ionescu

Dacă *încă* nu

EȘTI PREGĂTIT PENTRU MÂINE,

trebuie să mergi la ȘCOALĂ, *trebuie* să ÎNVEȚI, îți *trebuie* ... un MANUAL.



Prof. univ. dr. ***IOAN ODĂGESCU***, este absolvent al *Facultăților de Cibernetică, Statistică și Informatică Economică* (***Academia de Studii Economice***) și *Matematică-Mecanică* (***Universitatea București***).

Cu o vechime de peste 25 de ani în învățământul superior de Informatică Economică, membru al Catedrei de Informatică Economică din ASE, a publicat multe lucrări de referință pe diferite domenii, cum ar fi:

În domeniul analizei numerice:
METODE NUMERICE ȘI SUBRUTINE
OPTIMIZAREA ALGORITMILOR

În domeniul informaticii:
VIRUȘI INFORMATICE ȘI BOMBE LOGICE – prima lucrare de acest gen din țară
LIMBAJUL COBOL FĂRĂ PROFESOR
SIMULAREA PROCESELOR ECONOMICE, GRAFICA INTERACTIVĂ PE CALCULATOARE PERSONALE
METODE ȘI TEHNICI DE PROGRAMARE – enunțuri, soluții, probleme propuse <<cartea galbenă>>)

Trebuie remarcat faptul că, domnul profesor a coordonat colectivul care a publicat prima lucrare pe plan național, întitulată OLIMPIADE NAȚIONALE ȘI INTERNAȚIONALE DE INFORMATICĂ

Domnul prof. Ioan Odăgescu este conducător de doctorat în specialitatea Informatică Economică. În cadrul acțiunilor științifice de promovare a tinerelor talente, de aproximativ 20 de ani se ocupă cu organizarea concursurilor de informatică și este autor a multor probleme date la concurs.

Este membru în consiliul ***Societății Române de Informatică*** (SOCROM INFO) și membru ***ACM*** (USA).

* * *

Asist. univ. drd. ***FELIX FURTUNĂ*** este absolvent al *Facultății de Cibernetică, Statistică și Informatică Economică* (***Academia de Studii Economice***, București), specializarea *Informatică Economică*. În prezent este cadru didactic la *Facultatea de Cibernetică Statistică și Informmatică Economică*, catedra de Informatică Economică.

Este coautor al lucrărilor:
METODE ȘI TEHNICI DE PROGRAMARE; OLIMPIADE NAȚIONALE ȘI INTERNAȚIONALE DE INFORMATICĂ
GRAFICA INTERACTIVĂ PE CALCULATOARE PERSONALE

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422] sau la Computer Press AGORA [Tel: 065-166516; 01-3309281].

PROBLEME propuse pentru REZOLVARE

După câteva numere ale GInfo în care v-am "necăjit" cu probleme grele, vă propunem un set de enunțuri care au constituit proba regională pentru concursul "Nemes Tihamér", cu etapa finală la Budapesta.

P039901: Olimpiadă

La Olimpiada de informatică din Țara lui Papură Vodă se organizează baraje cât timp există concurenți care se află la egalitate. La fiecare baraj nou, participarea este obligatorie numai pentru acești concurenți, iar punctajul obținut se folosește numai pentru departajarea acestora. (După ultimul baraj nu mai există concurenți la egalitate.) La baraje pot participa și acei concurenți a căror clasare este deja stabilită. Aceștia vor primi puncte ca și ceilalți, dar nu-și mai pot schimba poziția în clasament. Dacă nu participă la un baraj, primesc 0 puncte.

Pe prima linie a fișierului **OLI.IN** se află scris numărul concurenților ($1 \le N \le 1000$) și numărul barajelor ($1 \le M \le 10$). Pe următoarele M linii sunt scrise rezultatele celor M baraje. Fiecare linie conține N numere nenegative despărțite prin câte un spațiu.

Scrieți un program care scrie în fișierul **OLI.OUT** numărul de ordine al candidaților, în ordine descrescătoare după punctajul obținut, despărțite prin câte un spațiu.

Exemplu:

OLI.IN:	OLI.OUT:
5 3 7 3 3 9 7 2 7 7 0 1 0 3 2 0 8	4 1 5 2 3

P039902: Familia

Relațiile familiale se definesc astfel încât corespunzător fiecărei persoane se precizează numele mamei și al tatălui.

Pe prima linie a fișierului **FAM.IN** se află numărul persoanelor cunoscute ($1 \le N \le 100$). Pe următoarele $3*N$ linii sunt scrise datele referitoare la cele N persoane: numele persoanei, numele tatălui și numele mamei. Ultima linie conține un singur nume care reprezintă numele persoanei ale cărei rude trebuie să le depistați. Un nume poate conține cel mult 20 de caractere.

Scrieți un program care va scrie în fișierul **FAM.OUT** următoarele 4 linii:

A. Numărul fraților persoanei precizate și numele lor, despărțite prin câte un spațiu.
B. Numărul fraților vitregi ai persoanei precizate și numele lor, despărțite prin câte un spațiu. (Frații vitregi au fie aceeași mamă, fie același tată.)
C. Lista strămoșilor pe linia tatălui persoanei precizate (tatăl, bunicul, străbunicul, stră-străbunicul etc., atât cât e posibil de depistat), despărțiți prin câte un spațiu.
D. Numărul verilor primari ai persoanei precizate și numele lor, despărțite prin câte un spațiu. (Verii au un bunic sau o bunică comună printre persoanele cunoscute.)

Exemplu:

FAM.IN:	FAM.OUT:
7 Pop Andrea Pop Stefan Rus Ana Pop Andrei Pop Stefan Rus Ana Pop Erica Pop Stefan Popescu Mihaela Pop Stefan Pop Petru Cretu Eleonora Rus Ana Rus Ioan Cuc Felicia Bucur Silvia Bucur Florin Cuc Felicia Cretu Melinda Cretu Dan Bucur Silvia Pop Andrea	1 Pop Andrei 1 Pop Erica 2 Pop Stefan Pop Petru 1 Cretu Melinda

P039903: Joc de cuvinte

Un joc logic de tip scrabble se joacă pe baza unui dicționar. Esența jocului constă în determinarea unei secvențe de cuvinte, pornind de la un cuvânt dat. Secvențapoate fi compusă doar din cuvinte existente în dicționar. Cuvinte consecutive se pot obține pe baza unei transformări elementare din cuvântul precedent. Există două reguli care trebuie respectate în efectuarea unei transformări elementare:

Regula A: schimbarea unei litere cu o altă literă;

Regula B: schimbarea unei litere cu o altă literă sau adăugarea unei litere la capătul cuvântului.

Pe prima linie a fișierului **DICTIO.IN** se află numărul cuvintelor din dicționar ($1 \le N \le 1000$), pe următoarele N linii avem câte un cuvânt din dicționar. Un cuvânt este format din cel mult 20 de caractere. Pe prima linie a fișierului **JOC.IN** se află litera regulii care trebuie aplicată (A sau B), iar pe următoarele două linii se află câte un cuvânt din dicționar (P și Q, cuvinte diferite).

Scrieți un program care scrie în fișierul **JOC.OUT** următoarele:

1. Pe prima linie se va scrie cuvântul DA în cazul în care din cuvântul P se poate obține cuvântul Q conform regulii date, altfel se va scrie NU. În caz afirmativ, după cuvântul DA va urma un spațiu, apoi un număr care reprezintă numărul minim de aplicații ale regulei în urma cărora se poate obține Q din P.
2. Pe cea de a doua linie se vor scrie toate cuvintele care se pot obține din cuvântul P pe baza regulei.

Exemplu:

DICTIO.IN:	JOC.IN:
7	A
OKOS	OKOS
ÁKOS	ÁLOM
ÁKOM	
OKOD	**JOC.OUT:**
ÁLOM	DA 3
ALOM	OKOS ÁKOS ÁKOM ÁLOM ALOM OKOD
HALOM	

P039904: Cuburi

Turnurile de cuburi se pot construi așezând deasupra unui cub cuburi mai ușoare și având laturi mai mici.

Scrieți un program care determină cel mai înalt turn care se poate construi din N cuburi! Înălțimea se calculează însumând laturile cuburilor din turn.

În prima linie a fișierului CUB.IN se află numărul cuburilor ($1 \le N \le 1000$). În următoarele N linii se află latura și greutatea cuburilor (numere naturale mai mici decât 20000) despărțite printr-un singur spațiu.

Să se scrie pe ecran și pe prima linie a fișierului **CUB.OUT** numărul M al cuburilor din care s-a construit cel mai înalt turn, următoarele M linii fiind rezervate laturilor și greutăților acestora în ordinea construcției (de jos în sus).

Exemplu:

CUB.IN:	CUB.OUT:
5	3
10 3	20 5
20 5	10 3
15 6	10 2
15 1	
10 2	

P039905: Prietenii

Într-un grup de N elevi s-a pus întrebarea: *Cine îți este cel mai simpatic?* Datele au fost trecute într-un tabel.

Pe prima linie a fișierului **PRIETEN.IN** se află numărul elevilor ($2 \le N \le 1000$), restul de N linii vor fi completate cu numele elevilor și cu răspunsul lor la întrebare, separate prin spații. Numele elevilor precum și răspunsul nu pot conține spații și nu pot fi mai lungi decât 20 de caractere.

Scrieți un program care scrie în fișierul **PRIETEN.OUT** (linie cu linie) următoarele:

A. În prima linie se va scrie numele celor care nu sunt simpatizați de nimeni, separate prin câte un spațiu.

B. În cea de a doua linie se vor scrie acele perechi care se simpatizează reciproc, dar nu sunt simpatici altora. Cele două nume, aparținând unei perechi se vor uni cu o liniuță (-), iar o pereche se va despărți de o altă pereche printr-un spațiu.

C. În cea de a treia linie se va scrie numele celui mai simpatizat elev. Dacă sunt mai mulți, numele lor se vor despărți printr-un spațiu.

D. În cea de a patra linie se va scrie numărul cel mai mare de grupuri în care pot fi împărțiți elevii, astfel încât fiecare elev să ajungă în acel grup în care este și persoana pe care o simpatizează cel mai mult.

Exemplu:

PRIETEN.IN:	PRIETEN.OUT:
5	B
A C	D-E
B A	A
C A	2
D E	
E D	

P039906: Proiect

O construcție de mare anvergură a început cu desemnarea pozițiilor posibile a clădirilor pe terenul de construcție. Toate clădirile au baza în formă de dreptunghi. Acesta se va preciza prin coordonatele colțului stânga-jos (x, y) și prin lungimile laturilor paralele cu axa *Ox* (dx), respectiv axa *Oy* (dy). Clădirile trebuie așezate în așa fel încât să nu se acopere, adică privind din origine nici o clădire nu poate acoperi nici un punct al altor clădiri.

Scrieți un program care determină numărul maxim de clădiri care nu se acoperă una pe alta.

Pe prima linie a fișierului **PLAN.IN** este scris numărul așezărilor posibile (1<N<5000). Următoarele N linii conțin fiecare câte patru numere naturale despărțite prin câte un spațiu, reprezentând datele corespunzătoare unei posibile așezări: x y dx dy (0<x,y,dx,dy<20000). Primele două numere corespund coordonatelor stânga jos ale clădirii (x,y), al treilea și al patrulea reprezintă lungimea laturii paralele cu axa *Ox*, respectiv cu axa *Oy*.

Pe prima linie a fișierului **PLAN.OUT** se va scrie numărul maxim (K) de clădiri care se pot așeza respectând cerințele problemei. Pe următoarele K linii se vor scrie datele referitoare la clădirile așezate, în aceeași structură ca în fișierul **PLAN.IN**.

Exemplu:

PLAN.IN:	PLAN.OUT:
9	4
3 11 2 3	1 8 2 4
1 8 2 4	6 10 2 2
4 2 1 1	4 2 1 1
6 10 2 2	11 1 3 2
6 6 1 3	
6 6 1 7	
7 9 2 3	
11 1 3 2	
7 4 2 1	

P039907: Camion

O întreprindere produce subansambluri în uzinele sale din diferite orașe ale țării. Producția realizată pe parcursul unei săptămâni este transportată cu camioane în depozitul central, la sfârșitul săptămânii. Din cauza limitării circulației, camioanele circulă între două orașe într-o singură direcție. De aceea întreprinderea planifică transporturile astfel încât din fiecare oraș în care nu se poate ajunge din alt oraș, pornește câte un camion, iar din celelalte nu pornește nici unul. Datorită restricțiilor, drumul fiecărui camion spre depozitul central este determinat în mod univoc.

Oricare camion al cărui drum trece printr-un oraș poate încărca oricâte subansambluri fabricate în acel oraș, cu condiția să nu fie plin. Cunoscând numărul subansamblurilor fabricate în fiecare oraș, determinați capacitatea minimă pentru camioane astfel încât transportul să fie realizat în întregime. Se presupune că toate camioanele au capacități identice.

Pe prima linie a fișierului **CAMION.IN** este scris numărul orașelor N (1<N<200). Depozitul central se află în orașul având numărul de ordine 1 și de aici nu se fac transporturi. Pe următoarele N-1 linii sunt scrise câte două numere întregi, despărțite prin câte un spațiu. Primul număr de pe linia I reprezintă orașul unde camionul poate ajunge din orașul I. Al doilea număr reprezintă numărul subansamblurilor fabricate în orașul I. (Drumul de ieșire din orașul 1 nu este dat.)

Programul va scrie în fișierul **CAMION.OUT** acea capacitate minimă a camioanelor care este suficientă pentru transportul tuturor subansamblurilor. Acest rezultat va fi un număr întreg.

Exemplu:

CAMION.IN	CAMION.OUT
8	12
1 2	
1 3	
1 4	
2 5	
3 6	
3 7	
4 8	

P029908: Pistă cu obstacole

Un vehicol se poate mișca pe un teren având structura unui caroiaj format din M linii și N coloane. Un pătrat din caroiaj poate fi vid sau poate conține o șină care îl traversează și pe care vehicolul se poate deplasa. Un pătrat are patru vecini în caroiaj, la stânga, la dreapta, în jos și în sus. Vehicolul poate efectua la un moment dat una din următoarele trei operații:

1. Învârte cu 90° pătratul pe care se află.
2. Trece pe un pătrat vecin dacă acesta conține o șină poziționată astfel încât șina de pe pătratul curent se potrivește cu ea.
3. Învârte cu 90° un pătrat vecin.

Scrieți un program care determină numărul minim al operațiilor pe care le va efectua vehicolul pentru traversarea pistei din colțul stânga sus (1,1) în colțul dreapta jos (M,N).

Pe prima linie a fișierului **PISTA.IN** se află o pereche de numere M N despărțite printr-un spațiu ($1 \le M, N \le 100$). Următoarele M linii conțin fiecare câte N numere despărțite prin câte un spațiu:

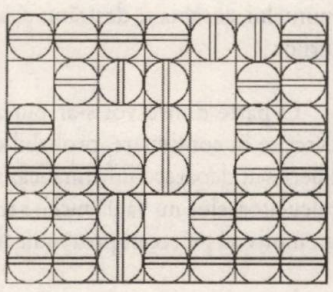
0: pătrățelul nu conține șină (este gol);
1: pătrățelul conține o șină orientată orizontal;
2: pătrățelul conține o șină orientată vertical.

Se va scrie în fișierul **PISTA.OUT** numărul minim al operațiilor pe care le va efectua vehicolul pentru traversarea pistei din colțul stânga sus în colțul dreapta jos. Dacă vehicolul nu poate traversa drumul, rezultatul va fi -1.

Exemplu:

PISTA.IN:

```
6 7
1 1 1 1 2 2 1
0 1 2 2 0 1 1
1 0 0 2 0 1 1
1 1 1 2 0 1 1
1 1 2 0 1 1 1
1 1 2 1 1 1 1
```



PISTA.OUT:

```
17
```

EFECTELE negative *ale calculatoarelor,* asupra noastră

Mihai Oltean

Stăm mult în fața calculatorului. Fie că lucrăm, fie că ne jucăm sau navigăm pe Internet. Facem bine? Nu facem bine?
Expunem în continuare câteva idei extrase din studii de specialitate efectuate de către instituții cu nume de prestigiu, cum ar fi MIT, WHO (World Health Organization) și ANSI (American National Standards Institute).

Pentru unii dintre noi, calculatorul este doar un simplu obiect ornamental. Așezat pe birou, arată bine, nu-i așa? Alții îl consideră o mașină extrem de utilă, care te ajută să realizezi calcule rapid. O altă parte dintre oameni văd în calculator un adevărat prieten, alături de care petreci clipe plăcute, te joci cu el, îi vorbești prin intermediul unui microfon, îl asculți când îți cântă melodia dorită. Dar oare nu tocmai din partea prietenilor primești lovituri care îți rănesc sufletul și care lasă urme greu vindecabile?

Imaginile viu colorate, calitatea excepțională a animației jocurilor, ușurința în utilizare, complexitatea aproape infinită a actualelor produse software, paginile WEB cu subiecte din cele mai variate domenii, toate sunt atrăgătoare, precum, pe vremuri, Muntele de Magnet atrăgea corăbiile care treceau pe lângă el. Ceea ce se întâmpla cu vapoarele respective și cu marinarii de pe ele, cred că intuiți: obiectele din fier erau smulse de puterea neobișnuită a muntelui, corabia se destrăma și pasagerii cădeau pradă valurilor.

O parte dintre voi s-ar putea să nu aibă simptomele descrise în continuare, probabil din cauză că sunteți prea tineri sau deoarece informatica, sau mai precis, utilizarea calculatoarelor nu vă dă nici o satisfacție, deci nu aveți nici un motiv să petreceți prea mult timp prin preajma lor.

Înainte de a începe expunerea propriu-zisă vreau să fac o analogie între utilizarea calculatoarelor și un alt obicei de care sunt sigur că toți ați uzat, și anume mersul pe bicicletă.

Închipuiți-vă că parcurgeți cu bicicleta o distanță scurtă, să zicem de maxim 1 km. Aceasta echivalează cu tastarea unui mic program, care să zicem, calculează suma a n numere, sau cel mai mare divizor comun al lor. Nu veți avea nici o problemă. Dar acum, imaginați-vă că nu mergeți doar 1 km, ci 20 km și nu doar pe un drum asfaltat ci și pe cărări pietruite și pline de gropi. La sfârșitul drumului veți simți dureri în tot corpul: la picioare, mâini, umeri, șolduri etc. Această călătorie echivalează cu tastarea unui program de câteva sute de linii. Veți simți dureri de ochi, de cap, ați amorțit pe scaun și vă găsiți într-o stare avansată de oboseală. După câteva ore de odihnă toate acestea vor trece și veți spune: *nici o problemă, totul e OK.* Dar, poate știți, poate nu știți, efectele negative asupra sănătății sunt cumulative: astăzi un pic, mâine un pic, și tot așa, până într-o dimineață, când... Dar să nu anticipăm, ci să luăm lucrurile pe rând.

Vom începe cu *efectele asupra fizicului*, iar apoi vom prezenta și *efectele asupra psihicului.*

Părțile cele mai afectate ale corpului sunt mâinile, mai precis degetele. Sute sau chiar mii de repetate și rapide apăsări ale tastelor nu pot trece fără urmări. Boala, sau mai precis grupul de boli generate, se numește ***R.S.I.*** (*Repetitive Strain Injuries*) sau în traducere *Vătămări prin Întinderi Repetate*. Simptomele sunt printre cele mai obișnuite și mai des întâlnite: mâini reci, disconfort, puțină durere, nevoia de a masa mâinile, diminuarea puterii de strângere.

Tratarea acestei boli durează poate câteva luni, iar ne-

tratarea ei duce mai târziu la tremuratul supărător și inestetic al degetelor, slăbirea forței de strângere și alte fenomene asemănătoare, datorate slăbirii rapide a mușchilor și tendoanelor uzate.

Durerile de cap și de ochi sunt datorate iluminării improprii, mai precis din cauza faptului că lumina provenită de la monitor este mai puternică decât cea a obiectelor din jur. Cred că mulți dintre voi, după o folosire de câteva ore a calculatorului, au început să observe pete de culoare roșie chiar și pe obiectele alb negru.

Alte efecte ar fi ochii uscați datorită faptului că pleoapele clipesc din ce în ce mai puțin și deci evaporarea se face mai repede.

Efectele negative asupra femeilor gravide sau influențarea apariției cancerului sunt, în prezent, doar presupuneri ce nu au fost confirmate deocamdată.

Toate aceste efecte negative pot fi înlăturate în mare măsură dacă locul de muncă este bine proiectat și pauzele sunt dese. Referitor la poziția corectă în fața calculatorului aș dori să spun că nu există ceva foarte bine definit și care să se poate aplica în toate situațiile. Oricum câteva indicații se pot da, și anume:

- încheieturile mâinilor trebuie să fie drepte, nu înclinate sau sprijinite pe masă;
- brațul trebuie să miște mâna astfel încât să nu apară necesitatea ca degetele să se întindă prea departe după taste;
- nu loviți tastele, ci atingeți-le ușor;
- folosiți ambele mâini pentru a efectua o operație precum CTRL-C sau ALT-F;
- mâinile și brațele trebuie ținute calde; mușchii și tendoanele reci sunt mult mai ușor supuse vătămărilor;
- nu folosiți calculatorul decât atunci când este necesar;
- dacă masa e prea înaltă atunci așezați tastatura pe genunchi;
- încercați să stați la o distanță cât mai mare de monitor, dar să nu fiți nevoiți, din această cauză, să vă aplecați înainte pentru a vedea ce scrie pe ecran;
- monitoarele trebuie așezate la 90 de grade față de geamuri astfel încât lumina soarelui să nu bată nici din fața și nici din spatele operatorului;
- chiar și o poziție perfectă poate genera dureri, dacă pauzele de relaxare nu sunt dese.

Următoarea problemă pe care o voi trata constă în *efectele calculatoarelor asupra psihicului*. Tot ceea ce urmează poate fi concentrat într-un singur cuvânt: ***dependență***.

Ce este aceea o dependență? Aceasta este o activitate sau o substanță care, folosită în mod repetat, degenerează într-un obicei, iar pentru care trebuie să plătim cu un preț sau cu o consecință negativă prea mare în raport cu beneficiile. Există o mare varietate de dependențe, una dintre cele minore fiind, de exemplu, obișnuința de a mânca un anumit tip de bomboane sau gravele dependențe de droguri și alcool. Poate nu credeți, dar și dependența de calculator este la fel de periculoasă ca și cea de alcool, ambele putând duce la destrămarea unei familii sau la efectele negative subliniate anterior.

Asistăm, în acești ani, la o perioadă în care oamenii petrec mult mai mult timp printre aparate, decât alături de alți oameni. Același efect a mai fost observat și în anii '30 când a apărut radioul și apoi prin anii '50 când a apărut televiziunea. Rezultă o dependență pe care marile firme producătoare de soft doresc să o facă tot mai puternică, și asta nu prin constrângeri, ci prin oferirea unor produse tot mai atractive cum ar fi spre exemplu jocurile sau site-urile Internet. Dintre consecințele acestei dependențe, aș enumera:

1. Pierderea noțiunii de timp după fiecare conectare.
2. Plimbări pe afară tot mai rare. Acest lucru poate provoca, în timp, scăderea rezistenței organismului în fața diferitelor boli.
3. Petrecerea a tot mai puțin timp pentru luarea mesei; eventual aceasta se face în fața monitorului.
4. Negarea faptului că petreceți prea mult timp în fața calculatorului.
5. Deschiderea cutiei de poștă electronică de mai multe ori pe zi.
6. Pornirea calculatorului sau conectarea la Internet când aveți încă foarte multe lucruri de altă natură de făcut.

Nu încercați să deveniți cel mai bun hacker, prețul plătit este prea mare!

Poate că ați auzit povestea soției care trimite un e-mail din bucătărie în sufragerie pentru a-și anunța soțul că masa e gata. Sau povestea persoanei care a scris o pagină de WEB pentru pisica sa, sau despre cea care tasta "com" după fiecare punct când folosea un editor de texte.

Concluzionând, aș dori să pun în balanță câștigurile și pierderile rezultate din folosirea calculatoarelor:

Avantaje:
- îmbunătățirea situației financiare
- învățarea a tot mai multe limbaje și medii de programare
- testarea a tot mai multor jocuri și vizitarea a tot mai multor site-uri Internet
- acumularea a tot mai multor cunoștințe

Dezavantaje:
- deteriorarea sănătății
- diminuarea productivității și a eficienței
- (atenție studenți!) eșecuri la examene
- pierderea noțiunii de scurgere a timpului
- distanțarea tot mai puternică de prieteni

Decizia vă aparține!

Mihai Oltean este student la Facultetea de Matematică, Secția Informatică a Universității Babeș-Bolyai din Cluj. Poate fi contactat prin e-mail la adresa: om7280@scs.ubbcluj.ro

BACALAUREAT 1999

V-am promis teze de BAC, dezvoltate pe subiecte, propuse în numărul precedent. Dacă nu am fi luat 10, scrieți-ne! ☺

Subiectul I

a) O înregistrare este o structură neomogenă a cărei componente, numite *câmpuri*, pot avea diferite tipuri.

b) Un pointer este o variabilă statică ce poate avea ca valori adrese ale unor variabile dinamice de un anumit tip.

Subiectul II:

a) Variabila `ART` este o înregistrare ale cărei câmpuri sunt: un șir de caractere (`Nume`), un întreg (`Varsta`) și un întreg lung (`Nr_tel`). În limbajul Pascal o variabilă de tip **`string`** ocupă 256 de octeți, un întreg ocupă 2 octeți, iar un întreg lung 4 octeți. În total avem 260 de octeți. În C++ un șir de 256 de caractere ocupă 256 de octeți, un întreg lung ocupă 2 octeți, iar un întreg lung 4 octeți. În total 260 de octeți. Pentru varianta C++ un răspuns corect este și 266 deoarece există compilatoare C++ pentru care întregii sunt reprezentați pe 4 octeți, iar întregii lungi pe 8 octeți.

b)Unul dintre cei 8 biți va fi folosit pentru semn, deci cel mai mare număr care poate fi reprezentat pe 8 biți (un octet) este $11111112 = 127_{10}$.

c) `2457 : 16 = 153 rest 9`
`153 : 16 = 9 rest 9`
`9 : 16 = 0 rest 9`
deci $2457_{10}=999_{16}$

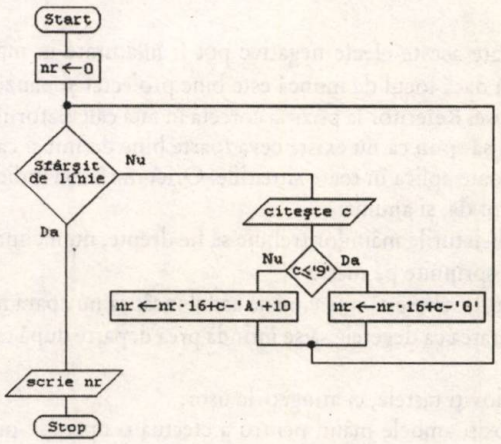
d) Un algoritm poate fi descris printr-o schemă logică, în pseudocod sau prezentând o implementare într-un limbaj de programare (în cazul nostru Pascal sau C++).

Iată cele 4 variante:

1. Pseudocod:

```
nr←0
cât_timp nu este sfârșit de linie execută
   Citește c
   dacă c <= '9'
   atunci nr ← nr·16+c-'0'
   altfel nr ← nr·16+c-'A'+10
sfârșit_dacă
sfârșit_cât_timp
scrie nr
```

2. Schemă logică:



Listing CONVERSIE.PAS

```
program Conversie_hexazecimal_zecimal;
var c:Char; nr:Longint;
Begin
  Write('Introduceti numarul hexazecimal: ');
  nr:=0;
  while not Eoln do
  begin
    Read(c);
    if c<='9'
    then nr:=nr*16+Ord(c)-Ord('0')
    else nr:=nr*16+Ord(c)-Ord('A')+10
  end;
  Writeln('Numarul zecimal: ',nr)
End.
```

Listing CONVERSIE.CPP

```
#include <stdio.h>
void main(void){
  long nr=0; char c;
  printf("Introduceti numarul hexazecimal:");
  c=GetChar();
  do{ if (c<='9') nr*=16,nr+=c-'0';
      else nr*=16,nr+=c-'A'+10;
      c=GetChar();
  } while (c!='\n');
  printf("Numarul zecimal: %ld\n",nr);
}
```

Subiectul III:

Se citește un șir de n numere, se inversează ordinea elementelor în șir și se afișează noul șir.

Inversarea elementelor șirului este realizată în structura repetitivă controlată de variabila i. În acest ciclu se realizează interschimbarea perechilor a_i și a_{n+1-i}. Se interschimbă primul element cu ultimul, al doilea cu penultimul etc. În continuare vom demonstra că cele trei instrucțiuni din cadrul ciclului realizează într-adevăr o interschimbare a elementelor. Notăm cu α și β valorile inițiale ale lui a_i, respectiv a_{n+1-i}. În urma primei instrucțiuni a_i va avea valoarea α+β, iar a_{n+1-i} va avea valoarea β. După a doua instrucțiune a_{n+1-i} va avea valoarea α+β-β=α, iar a_i va avea valoarea α+β. În final a_i va primi valoarea α+β-α=β, iar a_{n+1-i} va avea valoarea α. Deci, a_i va avea valoarea inițială a lui a_{n+1-i}, în timp ce a_{n+1-i} va avea valoarea inițială a lui a_i, ceea ce demonstrează faptul că cele trei instrucțiuni realizează o interschimbare.

Subiectul IV:

Pentru implementarea algoritmului de sortare vom scrie o procedură recursivă care, dacă este apelată pentru mai multe elemente, se va autoapela pentru șirul format din prima jumătate a elementelor și apoi pentru cea de-a doua jumătate, după care se vor interclasa cele două șiruri ordonate.

Procedura de interclasare creează cele două șiruri care trebuie interclasate, adaugă la sfârșitul lor câte o santinelă și apoi modifică șirul inițial astfel încât elementele care făceau parte din cele două șiruri să fie ordonate.

Deoarece se cerea scrierea unui program, trebuia prezentată o implementare în Pascal sau C++, nefiind acceptate scheme logice sau descrieri în pseudocod. Vă prezentăm în continuare versiunile Pascal și C++ ale implementării algoritmului (în care sperăm că recunoașteți o aplicație a metodei Divide et Impera):

Listing MSORT.PAS

```
program MergeSort;
var a,b,c:array[1..1000] of Integer;
    n:Integer;

procedure Citire;
var i:Integer;
begin
  Write('n='); ReadLn(n);
  Write('Numerele: ');
  for i:=1 to n do Read(a[i])
end;

procedure Inter(x,y:Integer);
var i,l1,l2:Integer;
begin
  for i:=x to (x+y) div 2 do b[i-x+1]:=a[i];
  b[i-x+2]:=MaxInt;
  for i:=(x+y) div 2 + 1 to y do
    c[i-(x+y) div 2]:=a[i];
  c[i-(x+y) div 2+1]:=MaxInt;
  l1:=1;
  l2:=1;
  for i:=1 to y-x+1 do
    if b[l1]<c[l2]
    then begin a[x+i-1]:=b[l1]; Inc(l1) end
    else begin a[x+i-1]:=c[l2]; Inc(l2) end
end;

procedure Sort(x,y:Integer);
begin
  if x<y then
  begin Sort(x,(x+y) div 2);
        Sort((x+y) div 2 + 1,y);
        Inter(x,y)
  end;
end;

procedure Scriere;
var i:Integer;
begin
  Write('Sirul ordonat este: ');
  for i:=1 to n do Write(a[i], ' ');
  WriteLn
end;

Begin
  Citire;
  Sort(1,n);
  Scriere
End.
```

Listing MSORT.CPP

```
#include <stdio.h>
#include <values.h>
int a[1000], b[1000], c[1000], n;
void citire(void);
void inter(int, int);
void sort(int, int);
void scriere(void);

void main(void){
  citire();
  sort(0,n-1);
  scriere();
}

void citire(void){
  printf("n=");
  scanf("%d",&n);
  printf("Numerele: ");
  for (int i=0; i<n;i++) scanf("%d",&a[i]);
}

void sort(int x, int y){
  if (x<y)
    sort(x,(x+y)/2),
    sort((x+y)/2+1,y),
    inter(x, y);
}

void inter(int x, int y){
  for (int i=x;i<=(x+y)/2;i++) b[i-x]=a[i];
  b[i-x]=MAXINT;
  for (i=(x+y)/2+1;i<=y;i++)
    c[i-(x+y)/2-1]=a[i];
  c[i-(x+y)/2-1]=MAXINT;
```

```
  int l1=0, l2=0;
  for (i=0;i<=y-x;i++)
    if (b[l1]<c[l2]) a[x+i]=b[l1++];
    else a[x+i]=c[l2++];
}

void scriere(void){
  printf("Sirul ordonat este: ");
  for (int i=0;i<n;i++) printf("%d ",a[i]);
  putchar('\n');
}
```

Subiectul V:

Acest subiect nu ar trebui să ridice probleme, crearea unei liste ordonate fiind prezentată în orice manual de Pascal sau C++ care se respectă, de aceea ne vom mulțumi să prezentăm doar implementarea algoritmului fără explicații suplimentare. Menționăm că am folosit o listă ordonată cu santinele.

Listing LIST_ORD.PAS

```
program Lista_ordonata;
type Lista=^element;
     Element=record
               val:Integer;
               leg:Lista
             end;
var f:Text;
    nr,i,n:Integer;
    cap,p:Lista;

procedure Init;
begin
  New(cap);
  New(p);
  p^.val:=MaxInt;
  p^.leg:=nil;
  cap^.leg:=p
end;

procedure Adauga;
var q:Lista;
begin
  p:=cap^.leg;
  while nr > p^.val do p:=p^.leg;
  New(q);
  q^:=p^;
  p^.val:=nr;
  p^.leg:=q
end;

procedure Afiseaza;
begin
  Assign(f,'OUTPUT.TXT');
  ReWrite(f);
  p:=cap^.leg;
  while p^.val<>MaxInt do
  begin
    Write(f,p^.val,' '); p:=p^.leg
  end;
  Close(f)
end;
```

```
Begin
  Init;
  Assign(f,'INPUT.TXT');
  Reset(f);
  ReadLn(f,n);
  Close(f);
  Assign(f,'NUMERE.TXT'); Reset(f);
  for i:=1 to n do
  begin Read(f,nr); Adauga end;
  Close(f);
  Afiseaza
End.
```

Listing LIST_ORD.CPP

```
#include <stdio.h>
#include <values.h>
struct lista{
  int val; lista *leg;
}*cap, *p;
FILE *f;
int nr;

void init(void);
void adauga(void);
void afiseaza(void);

void main(void){
  init();
  f=fopen("INPUT.TXT","RT");
  int n;
  fscanf(f,"%d",&n);
  fclose(f);
  f=fopen("NUMERE.TXT","RT");
  for (int i=0;i<n;i++)
    fscanf(f,"%d",&nr), adauga();
  fclose(f);
  afiseaza();
}

void init(void){
  cap=new lista;
  p=new lista;
  p->val=MAXINT;
  p->leg=NULL;
  cap->leg=p;
}

void adauga(void){
  p=cap->leg;
  while (nr>p->val) p=p->leg;
  lista *q = new lista;
  *q=*p;
  p->val=nr;
  p->leg=q;
}

void afiseaza(void){
  f = fopen("OUTPUT.TXT","WT");
  p=cap->leg;
  while (p->val!=MAXINT)
    fprintf(f,"%d ",p->val), p=p->leg;
  fclose(f);
}
```

BACALAUREAT 1999

Doina Rancea

V-am promis un Bac de info prin intermediul paginilor revistei. Rezolvați subiectele de mai jos, trimiteți rezolvările la redacție și vă vom spune dacă ați luat Bac-ul sau nu!

Subiectul I

Determinați ce calculează și afișează programul următor:

```
Program Bac;
var n,d:Word;
Begin
  Write('n='); Readln(n);
  d:=2;
  repeat
    while n mod d = 0 do
    begin
      Writeln(n:5,'|',d);
      n:=n div d
    end;
    d:=d+1
  until n=1
  Readln
End.
```

(0,75 puncte)

Subiectul II

Scrieți programul care implementează algoritmul următor, exprimat în pseudocod. Precizați funcția programului.

```
scrie 'n=';
citește n
repeta
  c ← rest[n/10]
  scrie c
  n ← [n/10]
până când n=0
stop
```

(0,75 puncte)

Subiectul III

Determinați prin metoda "divide et impera" minimul dintr-un șir de n (n<=50) numere întregi, introduse de la tastatură. Comentariile programului să evidențieze etapele metodei "divide et impera".

(1,5 puncte)

Subiectul IV

Să se creeze o listă de tip coadă, alocată dinamic, ale cărei elemente cuprind perechi de cuvinte (primul în limba română, al doilea este corespondentul său în engleză) citite de la tastatură. Lista trebuie să cuprindă numai cuvinte distincte. După creare lista se afișează. Crearea și listarea se fac fiecare într-un subprogram.

(2,5 puncte)

Subiectul V

Există creat pe discul C un fișier text numit **MEDII.TXT** ale cărui linii cuprind: numele, prenumele și media semestrială la programare a elevilor unei clase (câmpurile sunt separate printr-un singur spațiu). Elevii apar în fișier în ordine alfabetică. Să se afișeze lista elevilor promovați (în ordinea descrescătoare a mediilor), știind că nu există elevi neclasificați.

Observații:

1. Nu există elevi neclasificați.
2. Datele din fișier sunt corecte, nu necesită validare.

(1,5 puncte)

Subiectul VI

Directorul unui concern trebuie să inspecteze cele n (n<=10) întreprinderi ale concernului. Fiecare întreprindere se află în alt oraș. Pentru fiecare pereche formată din cele n orașe se cunosc costurile deplasării pe liniile aeriene. Să se determine traseul de inspecție având cost minim pe care trebuie să-l aleagă directorul.

Observații:

1. Orașele sunt identificate prin numere de la 1 la n.
2. Directorul se află în orașul 1. Aici trebuie să și revină.
3. Nu se admit soluții euristice.

(2 puncte)

Observație: Din oficiu se acordă 1 punct.

Doamna profesoară Doina Rancea este cadru didactic la Liceul de Informatică "Tiberiu Popoviciu" din Cluj; poate fi contactată prin e-mail la adresa: doina@licj.soroscj.ro

CALCULATOARE pe bază de ADN

Gheorghe Păun

Calculatoarele de astăzi sunt de milioane de ori mai puternice decât rudimentarii lor strămoşi din anii 40 sau 50. Aproape la fiecare 18 luni, ele devin de două ori mai rapide, în timp ce componentele lor devin de două ori mai reduse în volum. Sentimentul general este că tehnologia bazată pe circuite integrate se află foarte aproape de limitele sale. În ultimii ani, o întrebare aprins dezbătută a fost dacă nu trebuie să ne concentrăm asupra căutării unor calculatoare de un tip cu totul nou, de pildă, cuantice sau bazate pe "procesoare" moleculare. Vom prezenta pe scurt cea de-a doua direcţie menţionată mai sus, calculabilitatea pe bază de molecule organice, în special ADN.

Introducere

Proprietăţile care fac din ADN un posibil suport informaţional utilizabil *in vitro* sunt cunoscute de mai mulţi ani. O serie de operaţii asupra secvenţelor (simplu sau dublu-catenare) de ADN sunt din ce în ce mai bine stăpânite.

Speculaţii privind posibilitatea de a folosi ADN-ul ca mediu de calcul au fost făcute încă din anii '60.

În 1994, acest lucru a fost confirmat într-un anume sens: *L. Adleman* a publicat descrierea unui calcul (*găsirea unui drum Hamiltonian într-un graf*) efectuat în întregime prin manevrarea unor eprubete conţinând ADN! (Vom discuta acest experiment în secţiunea următoare.) La cel de-al doilea simpozion dedicat calculatoarelor pe bază de ADN (Princeton, iunie 1996), au fost comunicate şi alte experimente similare, cel mai complex dintre ele fiind deocamdată calcularea sumei a două numere în baza doi. S-a născut astfel un nou domeniu de cercetare, la confluenţa biochimiei cu informatica, având ca scop realizarea calculatorului molecular.

Bineînţeles, *miniaturizarea* nu este un scop în sine. Existenţa unor "procesoare" de dimensiuni moleculare face posibilă construirea unor calculatoare de mare paralelism, iar ADN-ul este un candidat extrem de promiţător: într-un volum mic încap miliarde de secvenţe de ADN, eventual în copii pe care să se execute în paralel anumite operaţii. Aceasta reprezintă trăsătura exploatată în mod esenţial de experimentul lui Adleman. Există şi alte calităţi ale posibilului calculator pe bază de ADN (consum energetic redus, posibilitatea de a reutiliza substanţele folosite, reversibilitatea unor reacţii) aşa cum există şi inconvenienţe previzibile (în special referitoare la nedeterminismul reacţiilor biochimice, deci la marja mare de eroare pe care trebuie s-o avem în vedere).

În acest moment cercetările din această arie sunt concentrate pe două direcţii principale: *găsirea unor aplicaţii cât mai convingătoare ale posibilităţii de calcul cu ajutorul ADN-ului* (deci de proiectare şi realizare a unor experimente efective de laborator care să rezolve probleme de interes real, eventual greu sau imposibil de rezolvat cu ajutorul calculatoarelor electronice) şi *investigarea teoretică, matematică, a puterii de calcul a eventualelor calculatoare cu ADN*. Din prima clasă fac parte o serie de proiecte spectaculoase, descrise pe scurt în diverse locuri, şi despre realizabilitatea cărora este greu să ne pronunţăm: "construirea unei memorii cu mult mai largi decât creierul uman" (*Science*, 268, Aprilie 1995), "spargerea sistemului criptografic *DES* (*Data Encryption Standard*, folosit de bănci şi de alte instituţii publice din SUA), lucrând trei luni în pivniţa casei" (la simpozionul organizat la Princeton, menţionat mai devreme, a fost prezentată o comunicare a unui colectiv din care făcea parte şi Adleman şi care cobora durata calculului la două luni...) şi aşa mai departe.

Vom avea în vedere cea de a doua direcţie de cercetare, cea de factură teoretică: cât se poate calcula, în raport cu ierarhiile de calculabilitate cunoscute, cu ajutorul unui model de calculabilitate bazat pe operaţii specifice ADN-ului? Se observă trecerea de la nivelul concret al experimentului de laborator la cel formal, matematic: se reţin

numai anumite caracteristici ale ADN-ului, numai anumite operații posibile sau doar plauzibile (unele doar dezirabile), și cu ajutorul acestor elemente se construiesc modele formale, "mașini" de calcul simbolic, ce sunt apoi comparate cu modelele de calcul clasice în teoria calculabilității, cum sunt mașinile Turing și variantele acestora. În afara faptului că preocupările autorului se referă la această direcție de cercetare, un motiv al restrângerii la această arie de interes este și faptul că în vederea construirii calculatorului molecular este necesară proiectarea sa în așa fel încât să-i asigurăm *universalitatea computațională*, faptul că el va fi echivalent ca putere de calcul cu calculatoarele existente. La nivel teoretic, acest lucru revine la echivalența cu mașina Turing. Pe de altă parte, deoarece raportat la cunoștințele actuale, fiecare model avut în vedere are caracteristici mai puțin sau deloc posibil de realizat practic, o strategie necesară este considerarea cât mai multor variante de asemenea modele, implicând diferite operații care eventual să fie implementate biochimic. Vom descrie, pe scurt, trei clase de modele.

Operațiile pe care se bazează aceste clase sunt:
- construirea unei secvențe dublu-catenare de ADN din porțiuni mono-catenare, complementare în sens ***Watson-Crick*** (A-T, C-G); în lipsa unui termen mai bun, vom numi această operație *polimerizare* (*matching*);
- *recombinarea* secvențelor de ADN sub influența enzimelor restrictive (*splicing* sau *cross-overing*);
- *inserarea și ștergerea* de simboluri sau de șiruri scurte, corespunzând mutațiilor genetice locale.

Aceste operații au fost identificate *in vitro* și în laborator sunt destul de bine controlate. (Scopul prezentării de față fiind familiarizarea cititorului cu problematica și rezultatele de natură informatică, plasate la un nivel idealizat, simbolic, nu vom intra în detalii.)

Pe baza operațiilor menționate anterior au fost construite trei tipuri de sisteme generative (pentru noi, "calcul" înseamnă generarea unui limbaj, a unei colecții de șiruri codificând o semnificație anume): le vom numi *P-sisteme*, *H-sisteme* și, respectiv, *I-sisteme* (*P* și *I* trimit la "polimerizare" și "inserare", *H-sistemele* sunt numite astfel în literatură pentru a aminti de ***T. Head***, cel care a formalizat operația de *splicing*, în 1987. Prezentarea nu va intra în amănunte tehnice (de exemplu, demonstrații). Cititorul interesat este rugat să consulte lucrările menționate la bibliografie pentru a găsi astfel de detalii.

O cuprinzătoare sursă bibliografică a abordărilor teoretice în această arie poate fi găsită în monografia *DNA Computing. New Computing Paradigms*, de ***Gh. Păun, G. Rozenberg, A. Salomaa***, publicată în 1998 la Springer-Verlag, Heidelberg.

Rezultatele sunt încurajatoare din punctul de vedere a calculabilității moleculare: *în toate cele trei cazuri se obțin mecanisme generative echivalente ca putere cu mașina Turing*. Mai mult, de cele mai multe ori se obțin *mecanisme generative universale*, programabile.

Evident, aceasta este doar o demonstrație teoretică a posibilității construirii de calculatoare universale bazate pe ADN. Fiecare dintre modelele considerate pleacă de la anumite presupuneri privind manevrarea secvențelor de ADN care cer progrese tehnologice esențiale, nefiind încă realizabile în laboratoarele actuale. Acest fapt ridică, pe de o parte, problema perfecționării posibilităților de operare asupra ADN-ului, iar pe de altă parte necesitatea căutării unor noi modele teoretice, combinând trăsături realiste ale modelelor existente, eventual bazate pe alte operații decât cele menționate. Ambele alternative cer lucrul în echipe interdisciplinare, dialogul dintre informaticianul-matematician și biochimist.

Experimentul lui Adleman

Nu avem aici de a face cu un calcul simbolic, ci cu unul analogic. Problema avută în vedere se referea la găsirea unui drum hamiltonian într-un graf, adică a unui drum care trece prin toate nodurile o singură dată. Ea a fost redusă la căutarea unor secvențe de ADN care codificau toate drumurile posibile în graf. Graful considerat a fost cel din fig.1.

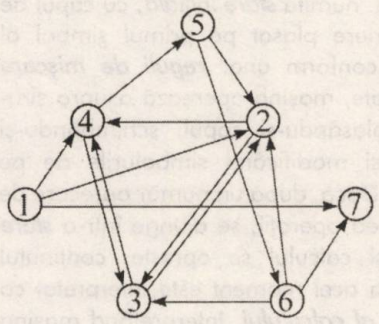


Fig. 1

Trebuie studiată existența unui drum hamiltonian în acest graf, plecând din nodul 1 și ajungând în nodul 7.

La o simplă examinare a grafului se poate afirma: drumul 1, 2, 3, 4, 5, 6, 7 este hamiltonian. Un calculator poate rezolva problema în câteva fracțiuni de secundă.

Problema face însă parte dintr-o clasă de probleme având complexitate exponențială. Mai precis, ea este NP-completă: nu se cunoaște nici un algoritm care s-o rezolve într-un număr de pași care depinde polinomial de numărul de noduri, cei mai buni algoritmi (determiniști) de rezolvare cer un timp exponențial, dar dacă o soluție este "ghicită" într-un mod sau altul, corectitudinea ei poate fi testată în timp polinomial (se spune că problema este NP-dificilă, NP prescurtând sintagma "nedeterminist polinomial"). Mai mult, orice problemă NP-dificilă poate fi redusă la problema noastră în timp polinomial (dacă problema ar putea fi rezolvată în timp polinomial, orice altă problemă NP-dificilă ar putea fi rezolvată astfel; acesta este înțelesul "completitudinii" din sintagma NP-complet). Pe scurt, problema găsirii drumurilor hamiltoniene între două noduri ale unui graf face parte dintr-o clasă de proble-

Mașina Turing

Conform tezei ***Turing-Church***, mașina Turing este cel mai general model de algoritm, orice alt algoritm putând fi simulat cu o mașină Turing.

Informal vorbind, o ***mașină Turing*** este:
- *un dispozitiv format dintr-o bandă potențial infinită, împărțită în celule în care pot fi scrise simboluri dintr-un alfabet precizat, un cap de citire-scriere și o memorie capabilă să se afle într-o stare dintr-o mulțime finită precizată;*
- capul de citire-scriere se poate mișca la stânga și la dreapta, citind simbolul din celula în dreptul căreia se găsește, eventual putându-l înlocui cu un alt simbol.

La începutul unui calcul, se scrie un șir de simboluri pe bandă. Plecând dintr-o stare specială, numită ***stare inițială***, cu capul de citire-scriere plasat pe primul simbol al șirului, conform unor ***reguli de mișcare*** specificate, mașina operează asupra șirului, deplasându-și capul, schimbându-și starea și modificând simbolurile de pe bandă. Dacă, după un număr oarecare de asemenea operații, se ajunge într-o ***stare finală*** și calculul se oprește, conținutul benzii în acel moment este interpretat ca ***rezultat al calculului***. Interpretând mașina ca generator de limbaje, putem spune că mașina a *recunoscut* șirul de intrare (ignorăm deci conținutul benzii la oprirea ei). Limbajele identificate astfel se numesc ***recursiv enumerabile*** iar familia lor se notează cu ***RE***.

Automat finit

O mașină Turing care își poate deplasa capul de citire numai spre dreapta, fără a ieși din limitele șirului scris inițial pe bandă și fără a putea modifica simbolurile de pe bandă, se numește ***automat finit***. Limbajele identificate astfel se numesc ***limbaje regulate***, iar familia lor se notează cu ***REG***.

Familiile ***REG*** și ***RE*** sunt limitele între care evaluăm puterea de calcul (generativă) a modelelor pe care le vom discuta.

me a căror rezolvare este dificilă (sunt numite probleme "netratabile"), indiferent de resursele de calcul folosite.

Se poate însă imagina un algoritm nedeterminist de rezolvare, care este implementabil cu ajutorul ADN-ului. Iată un asemenea algoritm:
1. generează la întâmplare drumuri în graf;
2. selectează dintre ele numai drumurile care pleacă din nodul inițial avut în vedere și se termină în nodul final considerat;
3. dacă graful are *n* noduri, reține doar drumurile care trec prin exact *n* noduri;
4. selectează drumurile care trec prin fiecare dintre nodurile grafului.

Drumurile rămase la sfârșitul celor patru pași (dacă mai există vreunul) sunt drumurile hamiltoniene căutate.

Acest algoritm a fost implementat de Adleman în felul următor (detalii pot fi găsite în [1]):

Fiecărui nod din graf i s-a asociat o secvență de 20 de nucleotide, generată la întâmplare. Lungimea a fost aleasă suficient de mare pentru a asigura stabilitatea secvențelor de ADN care codifică drumuri și pentru a face puțin probabilă existența unor porțiuni identice în codurile a două noduri diferite. Apoi au fost codificate arcele. De exemplu, dacă exisă un arc de la nodul *i* la nodul *j*, atunci el este reprezentat de o secvență de 20 de nucleotide, primele 10 fiind complementele *Watson-Crick* ale ultimelor 10 nucleotide din codul nodului *i* iar următoarele 10 fiind complementele primelor 10 nucleotide din codul nodului *j*. Din fiecare dintre aceste șiruri de nucleotide au fost introduse într-o eprubetă în jur de 3×10^13 copii, pentru a ridica probabilitatea ca orice drum din graf să fie realizat. În felul acesta, pasul 1 al algoritmului este executat, șirurile-arc se vor atașa la șirurile-nod, păstrând orientarea și codificând astfel drumurile în graf (fig.2).

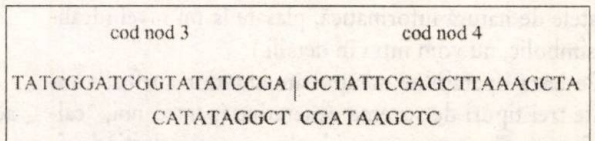


Fig.2

Pentru a asigura pasul al doilea al algoritmului, rezultatul primului pas a fost amplificat prin procedura *PCR* (*Polymerase Chain Reaction*), folosind ca *inițiatori de polimerizare* (*primers*) șirurile complementare codurilor nodurilor 1 și 7. Selectarea secvențelor de lungime 140 (avem 7 noduri, fiecare codificat printr-un șir de lungime 20) s-a făcut prin electroforeză pe gel. Prin diferite proceduri de purificare asupra cărora nu insistăm, a fost realizat și pasul al patrulea. În final, au fost găsite șiruri care atestau existența drumurilor hamiltoniene de la nodul 1 la nodul 7, iar prin "citirea" acestor șiruri au putut fi identificate precis aceste drumuri. Întregul "calcul" a durat în jur de o săptămână, aproape întreg timpul fiind consumat cu pașii 2-4 ai algoritmului. Acest lucru este semnificativ pentru dificultățile actuale ale lucrului cu ADN: principala problemă nu este introducerea datelor și nici calculul propriu-zis (aici, generarea tuturor

drumurilor în graf), ci extragerea rezultatelor, separarea secvențelor semnificative de cele nesemnificative. Pe de altă parte, complexitatea problemei rezultă tocmai din necesitatea generării și apoi a examinării tuturor drumurilor, hamiltoniene sau nu, iar acest pas a fost executat "automat", chimic, fără contribuția directă a omului.

Succesul lui Adleman a fost urmat atât de continuări entuziaste, cât și de evaluări pesimiste. De pildă, foarte curând procedeul a fost extins (a se vedea [8]) la alte probleme NP-complete, inclusiv problema SAT, a *satisfiabilității* formulelor de calculul propozițiilor (într-un anume sens problema NP-completă standard). Schema de lucru a lui ***Lipton*** a fost apoi preluată de Adleman, [2], care a propus un fel de *limbaj de programare* bazat pe operații de laborator asupra ADN-ului. În contrast, rămânând la problema drumurilor hamiltoniene, ***J. Hartmanis*** a calculat ([4]) că pentru a rezolva o problemă de interes real, de exemplu, referitoare la un graf cu 200 de noduri, ar fi nevoie de 3×10^{25} Kg de ADN, ceea ce este mai mult decât greutatea Pământului! Aceasta înseamnă că bariera exponențialității nu poate fi trecută ușor, și că algoritmul "naiv" folosit de Adleman nu este cel mai potrivit pentru a fi extins la probleme reale.

Rămâne însă demonstrația faptului că *putem realiza calcule folosind ADN-ul. Cât* putem calcula? Vom răspunde la această întrebare după ce vom stabili un sistem de referință. Aici, el va fi ierarhia lui ***Chomsky***, având ca limită superioară limbajele recursiv enumerabile, cele care pot fi "calculate" de mașina Turing, și ca limită inferioară limbajele regulate, cele recunoscute de automatul finit. Anticipând, vom spune că rămânând la nivelul operației de polimerizare din experimentul lui Adleman, nu putem depăși nivelul de competență al automatelor finite, ceea ce este foarte puțin; impunând anumite restricții, vom atinge puterea mașinii Turing. Părăsim astfel terenul complexității, al performanțelor calculatorului cu ADN, plasând discuția la nivelul competenței acestuia. Rezultatele teoretice sunt, spuneam, deosebit de promițătoare.

P-sisteme

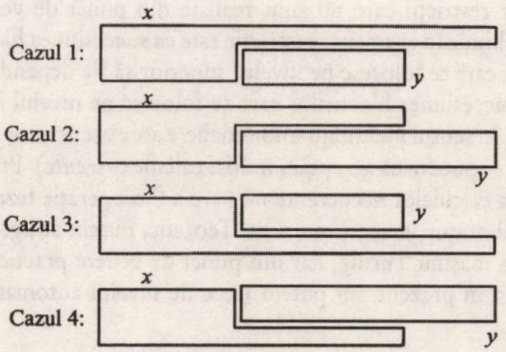
Aceste sisteme au fost introduse în [6] și ele pleacă de la operația de bază folosită de Adleman în experimentul său, aceea de construire a unei secvențe dublu-catenare de ADN folosind șiruri mono-catenare, complementare în sens ***Watson-Crick***.

Nu vom formaliza această operație, ci menționăm doar că ea este o funcție parțială, definită așa cum sugerează fig. 3:

În cazul 1 se adaugă simboluri complementare pe nivelul inferior, eventual fără a completa toate spațiile libere.

În cazul 2 se completează toate spațiile libere de pe nivelul inferior și eventual se mai adaugă și alte simboluri pe catena inferioară.

Cazurile 3 și 4 sunt simetrice, completând spații libere de pe nivelul superior al secvenței de simboluri etajate. De fiecare dată, completarea celor două niveluri se face sistematic, fără a lăsa spații libere, iar două simboluri pot sta unul deasupra celuilalt numai dacă ele sunt complementare, în sensul unei relații de complementaritate date (în cazul precis al ADN-ului, avem complementaritatea Watson-Crick; în general, pentru alfabete arbitrare, avem de stabilit în prealabil o relație de complementaritate între simboluri).



Mașini universale

Un calculator trebuie sa aibă o putere cât mai mare (cât mai apropiată de puterea mașinii Turing). Mai mult, este de dorit să avem calculatoare *programabile*, ceea ce corespunde ideii de universalitate:

- o mașină Turing M_U este ***universală*** dacă, pentru orice mașină Turing particulară M, putem construi un cod $w(M)$ astfel încât, plecând de la codul $w(M)$ și un șir x, mașina M_U lucrează exact așa cum lucrează mașina M asupra lui x.

Putem considera că $w(M)$ este un program care este executat de mașina M_U, pe datele reprezentate de x. Există mașini Turing universale, iar acest rezultat fundamental a fost demonstrat de ***Alan Turing*** încă din 1936. Pentru automatele finite nu există rezultate de universalitate, deci nu putem construi calculatoare programabile la acest nivel.

Aceste observații sunt fundamentale pentru calculabilitatea pe bază de ADN, unde trebuie deci să găsim modele de calcul de puterea mașinii Turing pentru a putea spera la construirea unui calculator (programabil).

Fig.3

Folosind această operație, putem defini un mecanism de calcul de genul următor: luăm un alfabet V, o relație de complementaritate ρ peste V, o mulțime finită de "molecule incomplete" de plecare (axiome), plus o mulțime finită de blocuri mono-catenare, pe care le folosim ca în fig. 3 pentru a completa secvențe dublu-catenare. Pornim de la axiome și prelungim spre dreapta secvența curentă cu blocuri uni-catenare. Ne vom opri atunci când obținem o secvență dublu-catenară completă. Spune că am calculat șirul obținut pe nivelul superior (sau putem considera o codificare adecvată a acestui șir). În felul acesta, putem asocia *P-sistemului* nostru un limbaj.

Există mai multe variante, atât ca formă a sistemelor folosite, cât și ca mod de lucru sau de oprire a calculului. Nu intrăm aici în amănunte tehnice. Important este că rezultatele obținute sunt de două tipuri: *caracterizări ale limbajelor regulate* (când operația de polimerizare este lăsată liberă) și *caracterizări ale limbajelor recursiv enumerabile* (când se impun anumite restricții asupra operației de polimerizare).

Prin urmare, modul de calcul din experimentul lui Adleman nu poate trece dincolo de competența de calcul a automatelor finite, ceea ce limitează foarte sever puterea unui "calculator" de acest tip. Pe de altă parte, atingerea puterii de calcul a mașinii Turing se face prin impunerea unor restricții care nu sunt realiste din punct de vedere biochimic (o asemenea restricție este ca succesiunea blocurilor care se folosesc pe nivelul superior să fie dependentă de succesiunea blocurilor care se folosesc pe nivelul inferior, în sensul identității unor etichete asociate acestor blocuri - spunem că acceptăm numai calcule *coerente*). Prevenirea calculelor necoerente nu pare a fi o operație fezabilă în laborator în acest moment. Teoretic, putem atinge puterea mașinii Turing, dar din punct de vedere practic, cel puțin în prezent, nu putem trece de nivelul automatului finit.

H-sisteme

În cazul în care două enzime restrictive taie două secvențe de ADN (în locuri precis definite de subsecvențe asociate celor două enzime și pe care acestea le recunosc în secvențele de ADN), dacă la capetele nou obținute se găsesc porțiuni complementare Watson-Crick, atunci fragmentele respective se pot recombina, dând eventual naștere unor noi secvențe de ADN. Modelul formal al acestei operații a fost introdus în [5] sub numele de *splicing*. Din acest punct de vedere, se poate spune că un pas important în calculabilitatea pe bază de ADN a fost făcut încă din 1987.

Să considerăm un alfabet V și două simboluri speciale, #, $, în afara lui V. O *regulă de recombinare* (peste V) este un șir $r=u_1\#u_2\$u_3\#u_4$, unde $u_i \in V^*$, $1 \le i \le 4$. Pentru o asemenea regulă r și două șiruri $x, y, z \in V^*$ definim

$$(x,y) \vdash_r z \Leftrightarrow \quad x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2$$
$$z = x_1u_1u_4y_2, \text{ pentru } x_1, x_2, y_1, y_2 \in V^*$$

Spunem că am recombinat șirurile x, y, la locurile u_1u_2, u_3u_4, obținând șirul z. Să remarcăm caracterul nedetreminist al operației: aceleași șiruri x, y pot da naștere mai multor șiruri z, în funcție de locurile șirurilor u_1u_2, u_3u_4 în x și y.

Operația de recombinare definită ca mai sus este ilustrată în fig. 4.

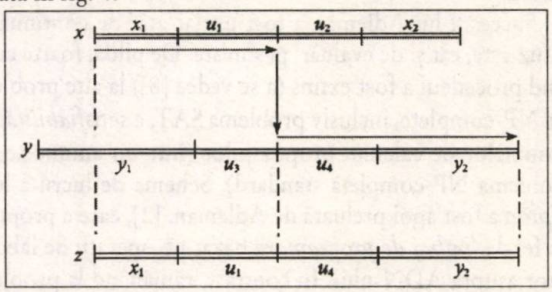


Fig. 4

O pereche $\sigma = (V,R)$, unde V este un alfabet, iar $R \subseteq V^*\#V^*\$V^*\#V^*$, se numește *H-schemă*. Pentru o H-schemă $\sigma = (V,R)$ și un limbaj $L \subseteq V^*$, definim

$$\sigma(L) = \{z \in V^* \mid (x,y) \mapsto_r z, \text{ pentru } x,y \in L, r \in R\}$$
$$\sigma^0(L) = L$$
$$\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),\ i \ge 0$$
$$\sigma^*(L) = \bigcup_{i \ge 0} \sigma^i(L)$$

În acest fel obținem două operații cu limbaje, σ (recombinare neiterată) și σ^* (recombinare iterată).

Pe baza operației iterate de splicing, în [11] a fost introdus un mecanism de calcul numit H-sistem, sub forma unui cuadruplu $\gamma = (V, T, A, R)$, unde V este un alfabet, $T \subseteq V$ (alfabet terminal), $A \subseteq V^*$ (mulțime de axiome) și $R \subseteq V^*\#V^*\$V^*\#V^*$, pentru #, $ simboluri în afara lui V; $\sigma = (V, R)$ este *H-schema subadiacentă* lui γ.

Limbajul generat de γ se definește prin:

$L(\gamma) = \sigma^*(A) \cap T^*$

pornind de la axiome, și aplicăm de un număr arbitrar de ori operația de recombinare în raport cu regulile din R și reținem șirurile formate numai din simboluri terminale.

Pentru două familii de limbaje, F_1, F_2, notăm cu $EH(F_1, F_2)$ familia limbajelor $L(\gamma)$, pentru $\gamma = (V, T, A, R)$ cu $A \in F_1$ și $R \in F_2$ (de notat că atât A cât și R sunt mulțimi de șiruri, deci limbaje, rezultă că are sens considerarea locului lor într-o ierarhie dată de limbaje, de pildă în ierarhia lui Chomsky).

Două rezultate fundamentale asupra acestor familii sunt:

Teorema 1. *Teorema de regularitate*

$$EH(FIN, FIN) = REG.$$

Teorema 2. *Teorema de universalitate*

$$EH(FIN, REG) = RE.$$

Incluziunea $EH(FIN, FIN) \subseteq REG$ este demonstrată în [3] iar incluziunea inversă este demonstrată în [11]. Teorema 2 este demonstrată în [9]. Prin urmare, simpla

trecere de la o mulțime finită de reguli de recombinare la o mulțime regulată (pasul cel mai mic care se poate face în condițiile considerate aici) asigură saltul de la familia *REG* la *RE*. Teoretic, rezultatul este spectaculos, dar din punct de vedere practic, a lucra cu o mulțime infinită de reguli de recombinare, fie ea și de tip regulat, nu prezintă un interes prea mare. Din demonstrația Teoremei 2 se poate observa că limbajul R folosit este de un tip foarte particular, iar infinitudinea lui poate fi evitată dacă operația de recombinare este restricționată în diferite moduri relativ simple.

De fapt, mai multe variante de H-sisteme extinse având mulțimi finite de axiome și de reguli de recombinare pot fi imaginate pe baza demonstrației din [9], toate conducând la caracterizarea familiei *RE*. Menționăm câteva dintre ele:

- *sisteme cu contexte permise* (*cp*): fiecare regulă este dată sub forma unui triplet $(r; C_1, C_2)$, unde C_1, C_2 sunt mulțimi de simboluri; o operație de recombinare $(x,y) \vdash_r z$ este permisă numai dacă toate simbolurile din C_1 apar în x și toate simbolurile din C_2 apar în y;
- *sisteme cu contexte interzise* (*ci*): regulile sunt ca mai sus și se aplică unei perechi de șiruri x, y atunci când nici un simbol din C_1 nu apare în x și nici un simbol din C_2 nu apare în y;
- *sisteme cu ținte locale* (*tl*): fiecare regulă este dată sub forma unei perechi (r,Q), unde Q este un limbaj regulat iar operația $(x, y) \vdash_r z$ este permisă numai dacă $z \in Q$;
- *sisteme cu țintă globală* (*tg*): ca mai sus, cu toate limbajele țintă asociate regulilor egale;
- *sisteme cu funcție de selecție* (*fs*): se dă o funcție $\varphi : V^* \to [0,1]$ (de fapt, este suficient să luăm $\varphi : V^* \to \{0, 1\}$) și permitem recombinarea $(x, y) \vdash_r z$ numai pentru șiruri x, y pentru care valorile $\varphi(x)$, $\varphi(y)$ sunt suficient de mari (la fel ca în cazul algoritmilor genetici).

Mai mult, din demonstrațiile faptului că H-sistemele cu operația de splicing restricționată ca mai sus caracterizează limbajele recursiv enumerabile, deducem existența H-sistemelor universale de tipurile menționate, adică a sistemelor γ_u cu toate componentele fixate și capabile să simuleze orice H-sistem dat γ, după adăugarea unui *cod* al sistemului particular γ la mulțimea de axiome a lui γ_u (prin urmare, γ poate fi "rulat" pe γ_u ca pe un calculator obișnuit, "programul" fiind o unică nouă axiomă adăugată "calculatorului" γ_u). Acest lucru este foarte încurajator din punct de vedere teoretic în ceea ce privește posibilitatea proiectării unor calculatoare pe bază de ADN care să fie universale și programabile.

Bineînțeles, dificultatea care apare aici este modul de realizare a unui asemenea calculator, de implementare a operațiilor menționate în discuția anterioară. Apar mai multe probleme.

1. Enzimele restrictive recunosc porțiuni de lungime redusă a secvențelor de ADN, de regulă în jur de 4-6. Acest fapt nu este important: reguli cu șirurile u_1u_2, u_3u_4 de lungime 4 sunt suficiente.
2. Enzimele cer condiții specifice de temperatură, salinitate etc., de aceea mai multe enzime nu pot lucra împreună decât cu mare dificultate. În demonstrațiile rezultatelor anterioare avem nevoie de un mare număr de enzime. Această dificultate este greu de surmontat. O posibilitate este considerarea de arhitecturi distribuite, ca în domeniul sistemelor gramaticale. Mai multe tipuri de *H-sisteme* distribuite similare sistemelor paralele și comunicante au fost considerate în numeroase lucrări. În toate cazurile se obțin caracterizări ale familiei *RE* și, pe baza demonstrațiilor rezultă existența sistemelor universale. Aceste tipuri de sisteme ridică însă alte dificultăți, în special legate de comunicarea între componente.
3. Controlul impus operației de recombinare este de inspirație teoretică, imitând restricții în derivare, uzuale gramaticilor independente de context. Implementarea acestor restricții în cazul *H-sistemelor* este o problemă deschisă.

I-sisteme

Este cunoscut faptul că ADN-ul este modificat nu numai prin recombinare, ci și prin mutații locale, în particular, prin inserare și ștergere de simboluri sau de șiruri de lungime limitată. Astfel de operații sunt bine-cunoscute în teoria limbajelor formale ([10]). Folosind asemenea operații pot fi definite mecanisme generative interesante; le vom prezenta aici în forma din [7].

Un *I-sistem* este o construcție $\gamma = (V, T, A, I, D)$, unde V este un alfabet, $T \subseteq V$ (alfabet terminal), A este o submulțime finită a lui V^* (axiome), iar I, D sunt submulțimi finite ale produsului $V^* \times V^* \times V^*$; I este mulțimea regulilor de inserare iar D este mulțimea regulilor de ștergere. O asemenea regulă se reprezintă sub forma (u, z, v).

Pentru $x, y \in V^*$ scriem $x \Rightarrow y$ dacă și numai dacă are loc unul dintre următoarele două cazuri:

- $x = x_1uvx_2$, $y = x_1uzvx_2$, pentru $x_1, x_2 \in V^*$ și $(u, z, v) \in I$ (pas de inserare);
- $x = x_1uzvx_2$, $y = x_1uvx_2$, pentru $x_1, x_2 \in V^*$ și $(u, z, v) \in D$ (pas de ștergere).

Notând ca de obicei cu $\Rightarrow^*$ închiderea reflexivă și tranzitivă a relației $\Rightarrow$, limbajul generat de γ se definește prin $L(\gamma) = \{w \in T^* \mid x \Rightarrow^* w$, pentru $x \in A\}$.

Spunem că un *I-sistem* $\gamma = (V, T, A, I, D)$ este de *grad* (n, m, p, q) dacă șiruri de lungime cel mult n sunt inserate în contexte de lungime cel mult m și șiruri de lungime cel mult p sunt șterse din contexte de lungime cel mult q.

Notăm cu $INS_n^m DEL_p^q$, $n, m, p, q \geq 0$, familia limbajelor $L(\gamma)$ generate de *I-sisteme* de grad (n', m', p', q') astfel încât $n' \leq n$, $m' \leq m$, $p' \leq p$, $q' \leq q$.

În [7] se demonstrează:

Teorema 3. $RE = INS_1^2 DEL_1^1 = INS_2^1 DEL_2^0 = INS_1^2 DEL_2^0$.

Prin urmare, folosind reguli de inserare și de ștergere de dimensiuni foarte mici (ca lungimi ale șirurilor inserate/șterse și ca lungimi ale contextelor în care aceste operații sunt efectuate), din nou putem atinge puterea mașinii Turing. Iar ca procedură de laborator prin care asemenea operații pot fi efectuate, putem imagina mai multe variante. Iată un mod de a insera șirul x în contextul (u, v), într-o secvență dublu catenară de ADN care are pe una dintre catene șirul $zuvw$. Prin încălzire, separăm cele două catene, apoi introducem în soluție complementul șirului uxv. Șirurile u și v se vor lega de șirurile complementare, lăsând șirul x ca o buclă în afară. Polimerizăm secvența obținută; în felul acesta, producem o secvență ADN dublu catenară care conține o buclă x într-o parte. Încălzim din nou soluția și separăm șirul $zuxvw$ de șirul complementar lui $zuvw$. Inserarea a fost realizată. Pentru a mări stabilitatea secvențelor rezultate după fiecare pas al acestei proceduri, contextul de inserare, (u, v) nu trebuie să fie prea scurt. Acest lucru face din limitele din Teorema 3 un rezultat mai interesant din punct de vedere matematic decât practic.

Atunci când avem de a face cu cazul concret al inserării/ștergerii în/din secvențe de ADN, lucrăm cu un număr limitat de simboluri, Se poate întâmpla ca unele dintre aceste simboluri să poată fi inserate sau șterse mai ușor decât altele. În particular, în cazul ARN-ului, inserarea și îndepărtarea uracilului (U) este mult mai simplă decât a celorlalte nucleotide.

Aceste observații fac interesant cazul *I-sistemelor* care au numai reguli de forma (u, z, v), cu $z \in c^*$ pentru un simbol precizat c. Asemenea *I-sisteme* au fost studiate în [12], dar nu intrăm în amănunte (în anumite condiții, ele generează din nou toate limbajele recursiv enumerabile).

Încheiere

Chiar dacă realizarea calculatorului (universal) pe bază de ADN nu este previzibilă pentru viitorul apropiat (principalele probleme se referă la erorile de calcul, la caracterul nedeterminist, probabilist al reacțiilor chimice), căutarea lui este o aventură incitantă atât pentru biolog cât și pentru informatician. Fiecare are de învățat de la celălalt și de reevaluat cunoștințe din propriul domeniu care își pot releva noi semnificații în noul context. Este de așteptat ca relativ curând să fie efectuate calcule specifice folosind proceduri biochimice (calcule ad-hoc pentru probleme ad-hoc), de exemplu, continuând experimentul lui Adleman. O arie care candidează cu mari șanse la aplicarea calculabilității pe bază de ADN este *criptografia* (unde ne putem permite o marjă mare de eroare a rezultatelor unui calcul, deoarece putem verifica aceste rezultate înainte de a le utiliza). În plus, nu este deloc lipsită de sens idea de a implementa modele de calcul sugerate de biochimia ADN-ului (H-sisteme, de exemplu), în medii nebiochimice, de pildă pe siliciu. Întrebarea este dacă un asemenea demers are o valoare practică sau nu, dar răspunsul nu poate fi dat înainte de examinarea atentă a acestei chestiuni. Oricum, din punct de vedere teoretic, calculabilitatea pe bază de ADN este deja bine dezvoltată iar rezultatele par deosebit de promițătoare. Cu siguranță, viitorul imediat va aduce mari progrese în această arie de cercetare.

Bibliografie

1. L. M. Adleman, *Molecular Computation of Solutions to Combinatorial Problems*, Science, 226 (Nov. 1994), 1021-1024.

2. L. M. Adleman, *On Constructing a Molecular Computer in DNA Computers. Procedings of a DIMACS Workshop* (R. J. Lipton, E. B. Baum, eds.), American Math. Soc., 1996, 1-21.

3. K. Culik II, T. Harju, *Splicing Semigroups of Dominoes and DNA*, Discrete Appl. Math., 31 (1991), 261-277.

4. J. Hartmanis, *On the Weight of Computations*, Bulletin EATCS, 55 (1995), 136-138.

5. T. Head, *Formal Language Theory and DNA: an Analysis of the Generative Capacity of Specific Recombinant Behaviors*, Bull. Math. Biology, 49 (1987), 737-759.

6. L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa, S. Yu, *DNA Computing, Sticker Systems, and Universality*, Acta Informatica, 35, 5 (1998), 401-420.

7. L. Kari, Gh. Păun, G. Thierrin, S. Yu, *Characterizing RE Languages by Insertion-Deletion Systems*, Proc. 3rd DIMACS Workshop on DNA Based Computers, Philadelphia, 1997, 318-333.

8. R. J. Lipton, *DNA Solution of Hard Computational Problems*, Science, 268 (April 1995), 542-545.

9. Gh. Păun, *Regular Extended H-Systems are Computationally Universal*, J. Aut., Languages, Combinatorics, 1, 1 (1996), 27-36.

10. Gh. Păun, *Marcus Contextual Grammars*, Kluwer, Dordrecht, Boston, 1997.

11. Gh. Păun, G. Rozenberg, A. Salomaa, *Computing by Splicing*, Theor. Computer Sci., 168 (1996), 321-336.

12. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Berlin, 1998.

Domnul prof. dr. Gheorghe Păun este membru corespondent al Academiei Române; poate fi contactat prin e-mail la adresa: gpaun@imar.ro

Arhitectura unei BAZE DE DATE

Ion Cozac

În prezent informația este o componentă esențială în desfășurarea oricărei activități. Deoarece influențează procesul de luare a deciziilor, informația trebuie să fie disponibilă în timp util, corectă, coerentă și neredundantă.
Cum satisfacem aceste cerințe în condițiile în care volumul datelor ce trebuie prelucrate este în continuă creștere? Evident, sunt necesare sisteme care să asigure culegerea, memorarea, organizarea, regăsirea și prelucrarea acestora. Aceste activități în informatică sunt legate nemijlocit de noțiunea de baze de date.

Acest articol deschide un serial care va prezenta fundamentele bazelor de date. Toate expunerile au un caracter general și nu fac referire la vreun SGBD particular. Vom încerca să dăm în același timp o notă practică tuturor expunerilor, aceasta pentru a ajuta viitorii proiectanți de aplicații de baze de date să găsească soluții cât mai bune pentru problemele care se pun. Structura cursului este următoarea:

1. *Arhitectura unei baze de date* (articolul prezent);
2. *Modelul relațional:* noțiunile de *domeniu*, *atribut*, *relație*, *cheie primară*, *cheie externă*; *probleme legate de integritatea datelor*; *modelul entitate-legătură*;
3. *Limbajul SQL* (partea I):
– descrierea datelor și prelucrarea acestora: *inserare*, *modificare*, *ștergere*;
– interogare – instrucțiunea `Select SQL`;
4. *Limbajul SQL* (partea a II-a):
– instrucțiunea `Select SQL` – selecții din mai multe tabele și funcții de agregare;
5. *Proiectarea bazelor de date:*
– dependențe funcționale și probleme cauzate de acestea;
– primele trei forme normale.

Informațiile prezentate în cadrul acestui curs sunt preluate dintr-o lucrare de referință în domeniu: ***Introduction to DataBase Systems***, autor *C. J. Date*, apărută la Addison-Wesley în 1994. Structura cursului este inspirată parțial din seria de articole care a apărut în primele 5 numere ale revistei PC Report (începând cu octombrie 1992) sub semnătura lui Mircea Sârbu. Informații detaliate despre fundamentele bazelor de date pot fi găsite la adresa http://www.uttgm.ro/~cozac, unde există un curs de baze de date preluat de la *Doublin City University*, și care urmărește în mare parte lucrarea lui *C. J. Date* amintită mai sus, precum și un tutorial pentru limbajul SQL.

Sperăm să răspundem astfel dorințelor exprimate de mai mulți elevi care doresc să afle informații ce nu se vor perima chiar atât de repede.

Avantajele bazelor de date

O ***bază de date*** este *o colecție de date persistente, utilizate de sistemul de aplicații al unei instituții.* ***Colecția de date*** este concretizată printr-un *ansamblu de fișiere pe disc care au o anumită stabilitate în timp, deci care nu sunt date de intrare, de ieșire sau de manevră.* Un ***sistem de gestiune a bazelor de date*** (în engleză ***DBMS: Data Base Management System***) este *un pachet unitar de programe care oferă facilități de descriere a datelor și de prelucrare a acestora.*

Aplicațiile de baze de date se caracterizează în primul rând prin faptul că majoritatea prelucrărilor care se fac sunt cele de memorare și regăsire a datelor, efectuate asupra unor volume mari de date. În general, operațiile de prelucrare sunt destul de simple, spre deosebire de alte domenii ale informaticii. De exemplu, în domeniul tehnic predomină operațiile de calcul cu o complexitate destul de ridicată. Cea mai frecventă operație care apare într-o aplicație de baze de date este aceea de ***consultare a datelor:*** într-adevăr, pentru ce creăm o bază de date, dacă nu o folosim?

Alte operații care apar pe lângă cea de consultare sunt:
– introducerea unor noi date;
– modificarea unor date existente;
– ștergerea unor date perimate.

Prin organizarea datelor în baze de date se asigură centralizarea acestora, fapt care conduce la o serie de avantaje:

1. *Reducerea redundanței datelor*
Dacă fiecare aplicație lucrează cu fișiere proprii este posibil ca aceleași date să apară de mai multe ori în fișiere diferite. În cazul centralizării datelor, administratorul bazei de date poate organiza datele, astfel încât toate aplicațiile să folosească aceleași fișiere. Astfel, se obține o economie importantă de spațiu de memorie, și nu doar atât.

2. *Evitarea inconsistenței datelor*
Duplicarea datelor în fișiere diferite poate crea probleme la actualizare: este posibil ca prin actualizări parțiale (din omisiune sau datorită unor accidente neprevăzute) să avem valori diferite pentru una și aceeași entitate (de exemplu, un client poate avea mai multe nume și nu mai știm care este cel real).

3. *Posibilitatea partajării datelor*
Se referă la posibilitatea utilizării datelor în comun de către mai mulți utilizatori și la posibilitatea dezvoltării de noi aplicații, folosind datele deja existente.

4. *Încurajarea utilizării unor standarde*
Administratorul bazei de date poate impune alinierea la anumite standarde, fapt care permite ulterior un transfer rapid al datelor de pe o platformă (hardware sau software) pe alta.

5. *Posibilitatea protejării datelor*
Administratorul bazei de date, având un control centralizat al datelor, poate introduce restricții diferite de acces la date pentru fiecare categorie de utilizatori.

6. *Menținerea integrității datelor*
Baza de date trebuie să conțină în permanență date corecte. Aceasta presupune date coerente și plauzibile, fapt care poate fi garantat de procedurile de validare utilizate.

7. *Independența datelor*
Într-o aplicație scrisă într-un limbaj clasic de programare, cunoștințele despre structura datelor și tehnicile de accesare a acestora sunt "zidite" în programe. Orice schimbare în modul de reprezentare sau accesare face imposibilă utilizarea aplicației: trebuie rescrise toate programele care referă aceste date. Independența datelor, garantată de utilizarea bazelor de date, presupune independența aplicației de modul de reprezentare a datelor și de tehnicile de acces utilizate.

Trebuie să facem o precizare: simplul fapt de a utiliza un SGBD în vogă la un moment dat, nu ne garantează automat obținerea acestor avantaje! Administratorul bazei de date trebuie să aibă o viziune de ansamblu asupra problemei care trebuie rezolvată, să cunoască toate datele problemei (datele de intrare, datele de ieșire, modul de prelucrare) și să cunoască facilitățile oferite de SGBD-ul folosit pentru a putea beneficia de avantajele de mai sus. În primul rând trebuie să aibă cunoștințe serioase despre proiectarea aplicațiilor de baze de date. Simplul fapt de a cunoaște "limbajul de programare Fox Pro" (fie el și vizual) – am dat un exemplu la modă – nu e suficient; pe de altă parte poate că uneori nici nu este necesar!

Arhitectura unei baze de date

Să luăm un exemplu simplificat din practică: ***evidența facturilor într-un magazin.***

În cazul în care un client dorește să cumpere niște produse, se întocmește o factură după modelul celei din fig. 1. Ceea ce este prezentat în figura respectivă este un *model extern* al bazei de date, care descrie modul în care vede vânzătorul datele acesteia. Dacă pe șeful magazinului îl interesează care produse se vând cel mai bine într-o anumită perioadă de timp, atunci el poate primi o situație după modelul din fig. 2.

Modelul extern este cel mai apropiat de utilizator și cuprinde descrierea structurii logice a datelor referitoare la aplicațiile fiecărui utilizator. Observăm că fiecare utilizator vede altfel baza de date, deși ea este unică și este memorată într-un mod ascuns tuturor.

Modelul conceptual cuprinde descrierea structurii datelor și a legăturilor dintre acestea pentru întreaga bază de date. În fig. 3 este prezentat modelul conceptual al bazei de date pentru problema evidenței facturilor. Baza de date se compune din patru tabele *(Clienti, Produse, Facturi, Detalii)* și pentru fiecare tabelă este prezentată structura

```
Număr factură: xxxxxxxxx                Data: xx.xx.xxxx
Cod client: xxxxxxxxx                   Nume: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Localitate: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Adresa: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Produse cumpărate:

Cod produs  Denumire                              Preț unitar  Cantitate   Valoare
xxxxxxxxx   xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx        xxxxxxxxx    xxxxxxxxx   xxxxxxxxx
yyyyyyyyy   yyyyyyyyyyyyyyyyyyyyyyyyyyyyyy        yyyyyyyyy    yyyyyyyyy   yyyyyyyyy

                                                  Total valoare:            zzzzzzzzz
```

Fig.1 Un model extern pentru vînzător

Perioada	Cod produs	Denumire	Preț unitar	Cantitate	Valoare totală
xxxxxxx	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxx	xxxxxxxxx	xxxxxxxxx	xxxxxxxxxxxx
yyyyyyy	yyyyyyyyy	yyyyyyyyyyyyyyyyyyyyy	yyyyyyyyy	yyyyyyyyy	yyyyyyyyyyyy

Fig.2 Un model extern pentru şeful magazinului

acesteia, cu alte cuvinte câmpurile care o descriu. Câmpurile cheie care servesc la identificarea unică a fiecărei înregistrări din tabelă sunt marcate cu (*). Astfel, pentru tabela *Clienti* cheia primară este `Codcl`, pentru *Produse*: `Codpr`, pentru *Facturi*: `Nrfact`; pentru tabela *Detalii* ea este formată din două câmpuri: `Nrfact` şi `Codpr`. Legăturile dintre tabelele componente se văd uşor, deoarece s-a folosit aceleaşi nume pentru câmpurile care asigură aceste legături.

Modelul conceptual este independent de modul de memorare a datelor, fapt care garantează independenţa datelor. Să remarcăm faptul că în fig. 3 nu se precizează tipul câmpurilor pentru nici o tabelă.

Clienti	Produse	Facturi	Detalii
* Codcl	* Codpr	* Nrfact	* Nrfact
Nume	Denumire	Dataf	* Codpr
Localit	Pretunit	Codcl	Cant
Adresa			

Figura 3. Modelul conceptual al bazei de date

Modelul intern defineşte modul de memorare a datelor şi tehnicile de accesare a acestora. La acest nivel se defineşte, pentru fiecare câmp, modul de reprezentare internă (dacă este şir de caractere, întreg binar, zecimal extern, boolean etc.). Baza de date apare ca o colecţie de fişiere având diferite moduri de organizare (liste înlănţuite, B-arbori etc.), fişierele conţinând datele care formează baza de date. Acest nivel este cel mai apropiat de sistemul de operare. Scopurile urmărite în alegerea modelului intern sunt *minimizarea spaţiului utilizat* şi *a timpilor de acces la date*. Despre *accesul la date* se pot citi câteva comentarii în articole publicate anul trecut despre tehnici de indexare bazate pe B-arbori şi hărţi de biţi.

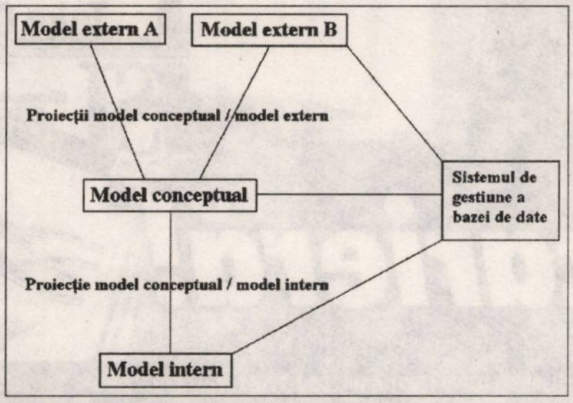


Fig.4 Arhitectura unei baze de date

În fig. 4 este prezentată o vedere de ansamblu a *arhitecturii unei baze de date*. De ce este oare necesară o arhitectură atât de complexă? Drumul pe care l-a parcurs activitatea de prelucrare a volumelor mari de date a fost destul de lung şi de sinuos. S-a urmărit o simplificare pe două planuri: pe de o parte accesul utilizatorului la date, pe de altă parte proiectarea aplicaţiilor.

Cum rezolvă SGBD-ul o cerere de acces pe care o face un utilizator? SGBD-ul interpretează această cerere în conformitate cu modelul extern al utilizatorului respectiv şi găseşte corespondenţa dintre modelul extern al acestuia şi modelul conceptual. În continuare determină componentele modelului conceptual care vor interveni în rezolvarea cererii şi face un acces la acestea pe baza corespondenţei dintre modelul conceptual şi modelul intern. După ce au fost identificate toate datele necesare, se revine din nou la modelul extern, acestea sunt integrate şi prezentate în final utilizatorului.

Trebuie să precizăm faptul că în practică modelul prezentat mai sus nu este întotdeauna respectat intregral. Multe SGBD-uri nu fac o distincţie netă între modelul conceptual şi cel intern. Scopul pe care l-am urmărit în expunerea de faţă este de a pune în evidenţă acea parte a bazei de date care, prin stabilitatea ei, garantează independenţa aplicaţiilor faţă de structura datelor şi coerenţa informaţiilor memorate în baza de date.

Vom vedea în articolele următoare (*prezentarea limbajului SQL*) faptul că atunci când se descrie o componentă a unei baze de date se precizează pentru fiecare câmp şi tipul acestuia. Acest lucru este firesc, deoarece atunci când proiectăm o bază de date avem în vedere şi operaţiile de prelucrare care se vor face. Dacă avem de efectuat operaţii aritmetice (adunări, scăderi, înmulţiri etc.), atunci operanzii trebuie să fie neapărat de tip numeric, ei nu pot fi şiruri de caractere!

Modelul conceptual este inima oricărei baze de date şi de aceea proiectarea acestuia este deosebit de importantă, deoarece de felul în care este rezolvată această problemă depinde soarta aplicaţiilor care vor exploata baza de date. Dacă acesta este atent proiectat, aplicaţiile se vor realiza relativ uşor şi vor fi fiabile. În caz contrar vor apărea mereu probleme pentru rezolvarea cărora va trebui să apelăm de multe ori la improvizaţii.

Administratorul bazei de date

O persoană sau un grup de persoane care răspunde de ansamblul activităţilor legate de baza de date se numeşte ***administrator al bazei de date***. Problemele pe care le rezolvă sunt următoarele:

– *decide care informații vor fi stocate în baza de date;*
Pe baza informațiilor pe care le primește de la viitorii beneficiari (utilizatori) ai bazei de date identifică ce entități prezintă interes pentru buna desfășurare a activităților instituției, și care dintre informațiile ce caracterizează aceste entități trebuie memorate. În acest moment el este în măsură să proiecteze ***schema conceptuală.***

– *decide structura datelor care vor fi memorate și tehnicile de accesare*
Administratorul bazei de date trebuie să urmărească stabilirea unui raport optim al spațiului de memorare/timpi de acces. Modelul intern este proiectat pe baza unei estimări a evoluției volumului de date, frecvenței accesării lor, a implicațiilor acestora în prelucrări.

– *proiectează modelele externe pentru fiecare categorie de utilizatori*
Administratorul bazei de date se informează în legătură cu cerințele acestora și eventual balansează cerințele conflictuale. Aceasta se realizează printr-o alegere atentă a modelului intern.

– *definește strategia de garantare a integrității datelor*
Există posibilitatea ca baza de date să nu mai fie operațională la un moment dat, din cauza unor erori în operare, a unor pene hard sau din alte cauze neprevăzute. Este foarte important pentru instituție ca baza de date să fie refăcută într-un timp cât mai scurt și cu pierderi minime de informații. În acest scop administratorul bazei de date elaborează strategii de salvare periodică a bazei de date și de refacere a acesteia (parțială sau în întregime) pe baza unor *copii de siguranță.*

– *monitorizează performanțele sistemului*
Pentru a obține o creștere a eficienței aplicațiilor urmărește modul de utilizare a resurselor și depistează factorii care limitează performanțele sistemului. El determină o planificare a funcționării sistemului pentru a maximiza performanțele acestuia.

Un set minimal de unelte pe care le folosește administratorul bazei de date în munca sa sunt:
– rutinele de inițializare (pentru încărcarea inițială a bazei de date);
– rutinele de import și export a datelor în și respectiv din baza de date (conversii);
– rutinele de jurnalizare și restaurare;
– rutinele de control și refacere a coerenței;
– rutinele de analiză statistică a funcționării sistemului.

Ion Cozac este lector la Universitatea "Petru Maior" din Târgu-Mureș și poate fi contactat prin e-mail la adresa: cozac@uttgm.ro

Cui nu-i place să deseneze?

MIRA - limbaj de programare pentru grafică

Marin Vlada

Grafica pe calculator (Computer Graphics) s-a dezvoltat și și-a conceput metodele și tehnicile cu care operează, simultan cu cercetările și rezultatele din domeniul informaticii, realizând o îmbinare între facilitățile oferite de software și facilitățile oferite de hardware. Prin intermediul metodelor și tehnicilor proprii și folosind software specializat, precum și dispozitive specializate, grafica pe calculator realizează procesarea obiectelor prin modelare, reprezentare, transformare și vizualizare.

Introducere

Preocupări privind obiectivele graficii pe calculator au apărut încă din deceniul 50, când în anul 1953 a fost conceput și construit primul *sistem grafic* ce utiliza un *display grafic* fabricat cu tub catodic cu un generator de vectori. În anul 1958, firma americană *CalComp* a fabricat primul *plotter* (masă de desen). *Ivan Sutherland*, prin teza sa de doctorat din 1963, a pus *bazele teoretice ale programării sistemelor grafice* și și-a creat o companie proprie de *echipamente grafice*. În anul 1968, firma americană *DEC* (*Digital Equipment Corporation*) a introdus în fabricație de serie primul *terminal grafic*.

Lipsa unor *modele* pentru *sistemele grafice* a condus la:

- dependența de *echipament*;
- dependența de *aplicație*;
- dependența de *domeniu* (mediu înconjurător).

Din aceste motive, pentru a obține *independența* față de elementele precizate mai sus, s-au conceput și elaborat *strategii de dezvoltare software* prin:

- realizarea de *limbaje de programare pentru grafică*, orientate spre programator;
- realizarea de *extensii grafice* (biblioteci) ale limbajelor de programare clasice, orientate spre programator;
- realizarea de *programe specializate de grafică*, orientate spre utilizator.

Până în prezent s-au proiectat și realizat o gamă diversificată de *produse-program* pentru grafica pe calculator:

- programe pentru *desenare automată* (desene tehnice, hărți, scheme etc.);
- programe pentru *generare și prelucrare de imagini*;
- programe pentru *tehnoredactare computerizată și publicistică*;
- programe pentru *animație* în 2D (bidimensional) și 3D (tridimensional);
- programe pentru *proiectare asistată de calculator* (*CAD-Computer Aided Design*);
- programe pentru *geometrie computerizată*;
- programe speciale de *umbrire* (*Rendering Software*);
- programe speciale de *iluminare* (*Reytracing Software*);
- programe pentru *prezentare și publicitate* (*Worksheet Graphics*).

Complexitatea și varietatea software-ului pentru grafica pe calculator a determinat și apariția *standardizărilor* în acest domeniu:

- standardizarea americană *CORE* (*CORE System*) - 1978;
- standardizarea germană *GKS* (*Graphical Kernel System*) - 1984;
- standardizarea americană *PHIGS* (*Programmer's Hierarchical Interactive Graphics Standard*) - 1984.

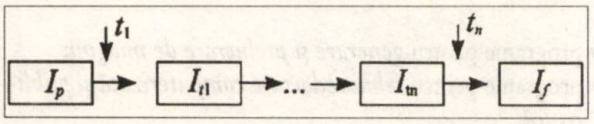
Un **sistem grafic** *este un ansamblu de* **componente hardware** *specializate pentru procesarea informațiilor grafice și* **componente software** *orientate spre domenii, spre programator sau spre utilizator.*

În comparație cu *sistemele de calcul numerice*, *cele grafice* au ca funcții de bază *memorarea, prelucrarea și citirea/scrierea informațiilor grafice*. Vom avea următorul paralelism:

Sisteme de calcul numerice	Sisteme grafice
- *informație alfanumerică*	- *informație grafică (codificare imagine/obiect)*
- *date numerice*	- *date grafice*
- *programe de calcul numeric*	- *programe de procesări grafice (modelare, reprezentare)*
- *baze de date*	- *baze de date grafice*

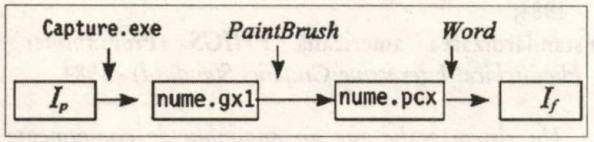
Componentele software ale sistemelor grafice trebuie să aibă implementate atât *metode de reprezentare*, cât și *metode de transformare* pentru procesarea informațiilor grafice. Aceste metode fac apel la concepte și noțiuni de matematică și de informatică. În procesul rezolvării problemelor, un rol important îl are *modelarea* pentru *reprezentarea* formelor ce apar și *transformarea* acestora în vederea obținerii rezultatelor cu ajutorul calculatorului.

În domeniul graficii computerizate, prin diversitatea programelor specializate și a domeniilor în care acestea sunt utilizate, au apărut tot mai multe metode pentru codificarea informației grafice (*formate grafice*). În procesul *modelării* se pornește de la o *informație primară* (I_p) din care prin intermediul unor *transformări* $t_1, t_2, \ldots, t_n$ se obține o *informație finală* (I_f) conform cerințelor problemei ce intră în procesul modelării:



unde I_{ti} sunt *reprezentări* intermediare.

Pentru exemplificare, vom considera că prin intermediul unui *program* scris în limbajul **Pascal** sau **C++**, pe monitor este realizată o *imagine* ce urmează a fi inserată într-un *document* prelucrat cu un program de tehnoredactare computerizată (**Word, Ventura**). În acest scop, imaginea realizată pe monitor prin execuția programului este capturată prin intermediul unui program ce trebuie să fie rezident în memorie și astfel se obține *un fișier*, de exemplu în format **GX1** (format grafic mai vechi), format recunoscut de editorul grafic **PaintBrush**. Prin utilizarea programului **PaintBrush**, fișierul obținut este convertit în format **PCX**, format ce poate fi importat de un program de tehnoredactare computerizată, de exemplu **Word**.



Limbaje și concepte OOP (Object Oriented Programming)

Fiecare mediu din natură este guvernat de diverse tipuri de limbaje. Acestea descriu *stări, acțiuni, forme, reprezentări, transformări* etc.

Un exemplu concludent privind diversitatea și complexitatea limbajelor îl oferă dezvoltarea și varietatea limbajelor utilizate pe diverse sisteme de calcul. Limbajele cunoscute până astăzi se încadrează în diferite niveluri. Dialogul Om-Calculator se realizează prin intermediul mai multor tipuri de limbaje prin care utilizatorul trebuie să ia cunoștință de *metodele de reprezentare*, *tehnicile de procesare* și în final de *modurile de interpretare*. În cele ce urmează, realizăm schematic o prezentare a diversității și complexității tipurilor de limbaje în relația Om-Calculator.

Paradigme ale limbajelor

Om - Calculator:

Limbaje

- Limbaje naturale
  (*entitate=cuvânt; descrie stări/ acțiuni/imagini*)
- Limbaje ale cunoașterii (*entitate=cunoștință*)
  - ♦ Limbaje științifice (matematică, fizică, chimie, informatică etc.)
  - ♦ Limbaje artificiale (*sisteme de calcul*)
    - ◊ Limbaje de programare clasică (*entitate= locație de memorie*)
    - ◊ Limbaje pentru baze de date (*entitate= înregistrare*)
    - ◊ Limbaje pentru grafică (*entitate=obiect grafic*)
    - ◊ Limbaje pentru modelare/simulare (*entitate = eveniment*)
    - ◊ Limbaje pentru sisteme de operare (*entitate= proces*)
    - ◊ Limbaje pentru Inteligența artificială (*entitate= cunoștință*)
  - ♦ Limbaj artificial (*sistem cibernetic*)
    - ◊ Reprezentare (*recepție*)
    - ◊ Procesare (*transformare*)
    - ◊ Interpretare (*emitere*)

Limbajele de programare moderne oferă diverse *tipuri de date* în scopul utilizării unor *structuri de date* adecvate complexelor probleme ce apar în proiectarea și elaborarea aplicațiilor pentru calculator. În general, un *tip de date* poate fi considerat ca fiind $T=(D,O)$, unde D este un *domeniu de valori*, iar O este o *mulțime de operații* definite pentru valorile din D.

Structura de date care stă la baza tehnicii OOP este `object`, gândită să *încapsuleze datele* (*proprietățile*) acestuia, dar și *metodele* (*procedurile*) de răspuns la stimuli (*evenimente, mesaje*). Tipul `object` (tipul `class` în C++) s-a obținut prin dezvoltarea *tipului procedural* împreună cu *tipul* `record` și prin intermediul *unităților de program* (`unit`). Tehnica OOP înglobează acele *metode specifice* realizării de programe complexe care sunt dirijate de un "*operator*" ce generează principalele "*evenimente*" ce urmează a fi prelucrate de către program.

Un model simplificat al *programării orientate pe obiecte* este:

```
while Non_Empty(Event_List) do
  ProcessCurrentEvent;
procedure ProcessCurrentEvent;
var event:EventType;
begin
  event:=Next_Event;
  case event of
    Event_1:procedure_1;
    ...
    Event_n:procedure_n
  end
end;
```

Se poate observa *asocierea* dintre *datele* `Event_i` și *procedura* `procedure_i`, procedură ce poate introduce *evenimente* noi în `Event_List`, sporind astfel considerabil complexitatea unui program scris în această filosofie. Impactul introducerii în limbajele de programare a tipului **object** (**class**) a permis realizarea și implementarea unor *biblioteci de obiecte*, de exemplu în Turbo Pascal:

- **objects** (*obiecte de bază Turbo Vision*) ;
- **menu** (*obiecte pentru lucrul cu meniuri*);
- **app** (*obiecte pentru generarea de aplicații Turbo Vision*).

Limbajul C++ pentru Windows este înzestrat cu *biblioteca OWL* (*Object Windows Library*) ce conține o ierarhie de clase create pentru *gestionarea prelucrării mesajelor* din diverse aplicații de tip Windows, principalele clase din aceasta ierarhie fiind `Object`, `Tmodule` și `TApplication`.

Se poate afirma că tehnica OOP este utilizată pentru *tratarea evenimentelor* (mesajelor) ce se nasc sau dispar pe timpul activității unei *aplicații (proces)*. Din acest punct de vedere programarea orientată spre obiecte gestionează *date și evenimente*.

Concepte OOP și limbajul MIRA

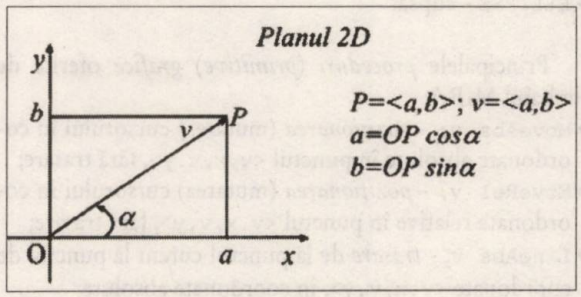
MIRA este un *limbaj de programare* pentru *grafică pe calculator* cu o sintaxă asemănătoare limbajului Turbo Pascal, dar care implementează *conceptele OOP* într-un mod natural și adecvat *obiectelor grafice* cu care operează. Dacă limbajele de programare clasice operează la nivelul *locațiilor de memorie* prin intermediul *tipurilor de date* și *procedurilor/funcțiilor*, iar limbajele de programare pentru bazele de date operează la nivelul *înregistrărilor* prin intermediul *bazelor de date* și a *procedurilor*, limbajul MIRA-limbaj de programare pentru grafică, operează la nivelul *obiectelor* prin intermediul conceptelor și tehnicilor OOP.

Acesta oferă prelucrări în 2D și 3D, de aceea consideră *planele* R^2 și R^3 ca fiind *spații vectoriale* peste R. Pentru comoditate, vom prezenta câteva elemente de grafică 2D oferită de MIRA. Orice ***punct*** *P* din R^2, determinat de coordonatele carteziene `(a,b)`, este definit de ***vectorul*** `v=<a,b>` legat de originea `(0,0)`. Conform limbajului MIRA, *vectorii* descriu *punctele* planului R^2.

Se consideră:

```
type vector=record x:Real; y:Real end;
var v,w:vector;
```

atunci `v.x` și `v.y` sunt *componentele* vectorului v, adică `v.x=projx(v), v.y=projy(v)`.



Vectorul *origine* este `origin=<0,0>`.

Distanța dintre două puncte v și w:

`dist(v,w)=SQRT((v.x-w.x)^2+(v.y-w.y)^2).`

Norma unui vector:

`norme(v)=dist(origin,v)=SQRT((v.x)^2+(v.y)^2).`

Vectorul unitar al lui v:

```
unitare(v)=<v.x/norme(v),v.y/norme(v)>=
                              =v/norme(v).
```

Observație: `norme(unitare(v))=1.`

Aritmetica vectorială (operații cu vectori):

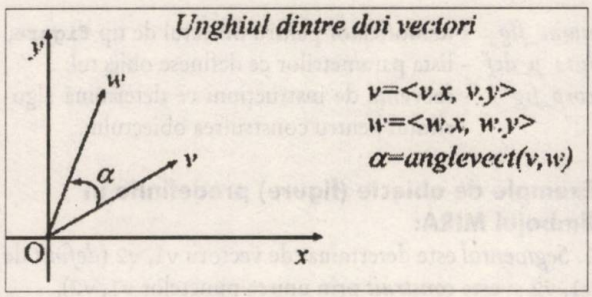
- *adunare* `v+w=<v.x+w.x,v.y+w.y>;`
- *scădere* `v-w=<v.x-w.x,v.y-w.y>;`
- *scalare* `r*v=<r*v.x,r*v.y>, r:Real;`
- *produs scalar* `v*w=v.x*w.x+v.y*w.y.`

Unghiul dintre doi vectori:

`anglevect(v,w)=arccos((v*w)/(norme(v)*norm(w))).`

Observație:

`v*w=norme(v)*norme(w)*cosα,`
`α= anglevect(v,w).`



Observație:

Vectorii v și w se numesc *ortogonali* dacă `v*w =0` și v diferit de `origin`; cu altă exprimare `cosα=0`, deci:

`α=anglevect(v,w)=π/2.`

Conceptul de figură (figure) în limbajul MIRA

Limbajul *MIRA* definește și operează cu *obiecte grafice* predefinite numite *figuri* (**figure**):

- SEG - segment de dreapta;
- TRIANGLE - triunghi;
- SQUARE - pătrat;
- CIRCLE - cerc;
- ELLIPSE - elipsă.

Principalele *proceduri (primitive) grafice* oferite de limbajul MIRA:

- MoveAbs v; - *poziționarea* (mutarea) cursorului în coordonate absolute în punctul <v.x,v.y>, fără trasare;
- MoveRel v; - *poziționarea* (mutarea) cursorului în coordonate relative în punctul <v.x,v.y>, fără trasare;
- LineAbs v; - *trasare* de la punctul curent la punctul de coordonate <v.x,v.y>, în coordonate absolute;
- LineRel v; - *trasare* de la punctul curent la punctul de coordonate <v.x,v.y>, în coordonate relative.

Observație:

Din punct de vedere sintactic, în limbajul MIRA *apelul* șirului de proceduri

```
LineAbs v1; LineAbs v2; ...; LineAbs vn;
```

este echivalent cu *apelul*

```
LineAbs v1; v2; ...; vn;
```

Conceptul de **figure** este utilizat în limbajul MIRA în conformitate cu tehnica OOP, pentru *definirea, memorarea, construirea (reprezentarea) și manipularea* unor porțiuni ale unei imagini.

Definirea unui obiect (**figure**) se face în secțiunea **type** și trebuie să conțină *caracteristici* ale obiectului prin parametrizare, respectiv *algoritmul* pentru construirea (reprezentarea) obiectului:

```
type nume_fig=figure(lista_p_def);
declarații
begin
  corp_fig
end;
```

nume_fig - identificator pentru obiectul de tip **figure**;
lista_p_def - lista parametrilor ce definesc obiectul;
corp_fig - secvența de instrucțiuni ce determină algoritmul pentru construirea obiectului.

Exemple de obiecte (figure) predefinite în limbajul MIRA:

1. *Segmentul* este determinat de vectorii v1, v2 (*definit* de v1, v2 și este *construit* prin unirea punctelor v1, v2).

```
Seg=figure(v1,v2:vector)
begin
  MoveAbs v1; LineAbs v2
end;
```

2. *Triunghiul* este determinat de vectorii v1, v2, v3 (*definit* de v1, v2, v3 și este *construit* prin unirea punctelor v1, v2, v3).

```
Triangle=figure(v1,v2,v3:vector)
begin
  MoveAbs v1; LineAbs v2;v3;v1
end;
```

3. *Cercul* este *definit* de centrul C și raza r și este *construit* prin unirea punctelor v=<r·cosα, r·sinα>, unde α=2kπ/n, k=0..n, iar n=100 este pasul de discretizare.

```
Circle=figure(C:vector; r:Real);
const n=100;
var pas,alfa:Real;
begin
  alfa:=0; pas:=2*pi/n;
  MoveAbs C+<r,0>;
  while alfa<2*pi do
  begin
    alfa:=alfa+pas;
    LineAbs C+r*<cos(alfa),sin(alfa)>
  end
end;
```

4. *Elipsa* este *definită* de centrul C, semiaxele a, b și este *construită* prin unirea punctelor v=<a·cosα, b·sinα>, unde α=2kπ/n, k=0..n, iar n=100 este pasul de discretizare.

```
type Ellipse=figure(C:vector; a,b:Real);
const n=100;
var pas,alfa:Real;
begin
  alfa:=0; pas:=2*pi/n;
  MoveAbs C+<a,0>;
  while (alfa<2*pi) do
  begin
    alfa:=alfa+pas;
    LineAbs C+<a*cos(alfa),b*sin(alfa)>;
  end
end;
```

Variabile de tip figură (figure)

O *variabilă* de tip **figure** este un obiect în sensul tehnicii OOP și suportă următoarele operații:

- CREATE - *crearea* obiectului (figurii) în memorie;
- DELETE - *ștergerea* obiectului (figurii) din memorie;
- DRAW - *desenarea* (afișarea-reprezentarea) obiectului pe dispozitivul de ieșire;
- ERASE - *eliminarea* (ștergerea) obiectului de pe dispozitivul (grafic) de ieșire.

Exemple privind *declararea* utilizării unor *variabile* de tip **figure**:

```
type info=record
           n:Integer;
           T:triangle;
           E:ellipse
         end;
```

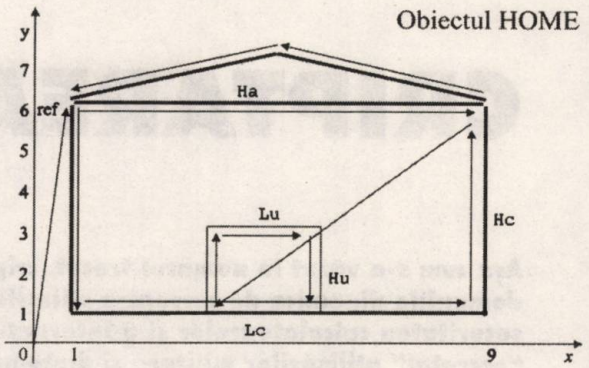
```
var V1,V2,V3:vector;
    T1,T2:triangle;
    C1,C2:circle;
    E1,E2:ellipse;
    VT:array [1..10] of triangle;
    C:array [1..10] of circle;
    VE:array [1.5] of ellipse;
    INR:info;
```

Exemple privind *utilizarea* unor *variabile* de tip **figure**:

```
...
CREATE T1(V1,V2,V3);
   // în memorie se creează triunghiul T1 definit de
   // vectorii V1, V2, V3.
CREATE C1(<2,3>,5);
   // în memorie se creează cercul de centru <2,3> și rază 5
CREATE C2(<4,5>,5);
   // în memorie se creează cercul de centru <4,5> și rază 5
CREATE E1(<3,5>,4,9);
   // în memorie se creează elipsa de centru <3,5> și
   //semiaxe 4, 5
for i:=1 to 10 do CREATE C[i](origin,i);
   // în memorie se creează un vector cu componente
   // cercuri cu centrul în origine și raze, respectiv i=1..10
...
DRAW T1,C1,C2,C[5];
   // se desenează pe dispozitivul grafic triunghiul T1
   // și cercurile C1, C2, C[5]
DRAW E1;
   // se desenează pe dispozitivul grafic elipsa E1
for i:=1 to 4 do DRAW C[i];
   // se desenează pe dispozitivul grafic cercurile
   // C[i], i=1..4
ERASE T1, C2, C[3];
   // se elimină de pe dispozitivul grafic T1, C2, C[3]
for i:=6 to 10 do DRAW C[i];
   // se desenează pe dispozitivul grafic cercurile
   // C[i], i=6..10
ERASE E1, C1, C[7];
   // se elimină de pe dispozitivul grafic E1, C1, C[7]
```

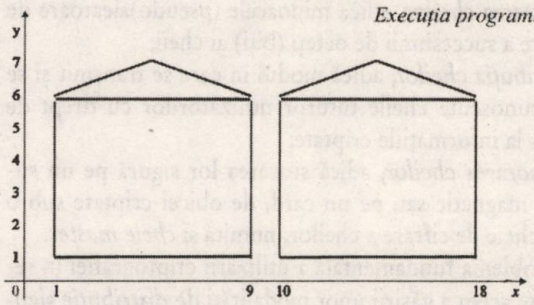
Aplicație

Vom elabora un *program* în limbajul MIRA în care vom defini un *obiect* (**figure** HOME) prin precizarea *parametrilor de definire* și a *algoritmului* pentru construirea obiectului. Schematic, vom defini o casă determinată de vectorul de referință `ref` și de dimensiunile: `Lc,Hc,Ha,Lu,Hu`, reprezentând *lățimea* și *înălțimea* casei, *înălțimea acoperișului, lățimea* și *înălțimea ușii*. În figura alăturată se prezintă forma obiectului ce se definește. Săgețile cu corpul subțire indică ordinea în care sunt *desenate* pe dispozitivul grafic segmentele ce constituie obiectul. *Construirea* obiectului se realizează prin apelul *procedurilor grafice* `MoveAbs` și `LineRel`, iar *desenarea* celor două obiecte se realizează prin apelul procedurilor `CREATE` (crearea în memorie) și `DRAW` (reprezentarea pe dispozitivul grafic).




```
Program HOME;
type
  HOME=figure(ref:vector;Lc,Hc,Ha,Lu,Hu:Real);
  var V1,V2,V3,V4,V5,V6;
      Sa,Sc:Real;
  begin
    Sc:=Lc/2; Sa:=Lu/2;
    V1:=<Lc,0>; V2:=<-Sc,Ha>; V3:=<-Sc,-Ha>;
    V4:=<0,-Hc>; V5:=<Lc,0>; V6:=<0,Hc>;
    MoveAbs ref;
    LineRel V1, V2, V3, V4, V4, V6;
    MoveRel <-Sc-Sa,-Hc>;
    V1:=<0,Hu>; V2:=<Lu,0>; V3:=<0,Hu>;
    LineRel V1, V2, V3
  end;

var Home1,Home2:HOME;
Begin
  CREATE Home1(<1,6>,8,5,1,2,2);
  CREATE Home2(<10,6>,8,5,1,2,2);
  DRAW Home1; DRAW Home2
     {desenarea a doua obiecte pe dispozitiv}
End.
```

Execuția programului



Domnul lect. dr. Marin Vlada este cadru didactic la Universitatea București; poate fi contactat prin e-mail la adresa: vlada@math.math.unibuc.ro

CRIPTAREA simetrică

Victor-Valeriu Patriciu

Așa cum s-a văzut în numărul trecut, criptografia a devenit azi unul dintre domeniile dinamice de cercetare științifică datorită aplicațiilor ei în securitatea calculatoarelor și a Internet-ului. Acoperită multă vreme sub "secretul" utilizărilor militare și diplomatice, astăzi criptografia trebuie "demistificată", fiind, nici mai mult nici mai puțin decât un instrument în activitățile informatice curente. În țările dezvoltate, se poate vorbi de o criptografie de uz cotidian.

Sisteme criptografice simetrice

În acest număr vom încerca să explicăm câteva concepte care stau la baza criptografiei moderne de tip simetric. ***Sistemele criptografice simetrice*** folosesc aceeași cheie *K*, atât la cifrarea cât și la descifrarea mesajelor. Cheia este secretă și este folosită în comun de către emițător, cel care cifrează mesajul *M* și de către receptor, cel care descifrează criptograma *C*.

Sistemele simetrice sunt folosite pentru protecția datelor memorate în fișiere, baze de date sau a informațiilor transmise în rețele. Criptosisteme simetrice ca standardul american de cifrare *DES* (*Data Encryption Standard*) sau sistemul *IDEA* (*International Data Encryption Algorithm*), sisteme cu o largă utilizare în protecția datelor în Internet, sunt bine cunoscute deja.

Securitatea criptării simetrice depinde mult de ***protecția cheii criptografice***. Ca urmare, administrarea acestora este un factor esențial și se referă la:

- ***generarea cheilor***, adică mijloacele (pseudo)aleatoare de creare a succesiunii de octeți (biți) ai cheii;
- ***distribuția cheilor***, adică modul în care se transmit și se fac cunoscute cheile tuturor utilizatorilor cu drept de acces la informațiile criptate;
- ***memorarea cheilor***, adică stocarea lor sigură pe un suport magnetic sau pe un card, de obicei criptate sub o altă cheie de cifrare a cheilor, numită și *cheie master*.

Problema fundamentală a utilizării criptografiei în rețele este aceea a găsirii unor modalități de ***distribuție sigură*** și periodică a cheilor criptografice, fiind necesar ca acestea să fie schimbate cât mai des. În Internet, pentru aceasta, se utilizează tot serviciile rețelei, folosind protocoale speciale sau sisteme cu chei publice, așa numitele ***anvelope (plicuri) digitale***.

Un exemplu de cifru simetric

Cel mai celebru cifru simetric, *DES* (*Data Encryption Standard*) are deja peste 20 de ani. El este primul standard dedicat protecției criptografice a datelor de calculator. A fost studiat de IBM începând cu 1970 pentru *NBS* (*National Bureau of Standards*). După câteva modificări realizate de *NBS* și *NSA* (*National Security Agency*), a fost publicat în 1977 și intitulat *DES*. Ulterior a fost adoptat de *ANSI* (*American Standard Institute*) ca standard *ANSI* X3.92 și intitulat *DEA* (*Data Encryption Algorithm*).

Privit în ansamblu, algoritmul DES nu este altceva decât o procesare ce urmărește "ascunderea" mesajului prin două căi: confuzie și difuzie. Construcția fundamentală a unui bloc DES este o combinație a unor tehnici - *substituție* urmată de o *permutare* - executate asupra textului, bazate pe o cheie. Această construcție este cunoscută ca *rundă*. DES este compus din 16 runde: aceeași combinație de tehnici se aplică de 16 ori asupra blocului de text clar. Algoritmul este compus dintr-un set de permutări, substituții și sumă *mod* 2, aplicate iterativ de 16 ori, pe un bloc de date de 64 biți, prin folosirea în fiecare rundă a unei chei diferite de 48 de biți, extrase inițial dintr-o cheie furnizată de utilizator de 56 biți; datele sunt împărțite în blocuri de 64 de biți și criptate fără a se modifica lungimea lor.

Progresele tehnologice din cei peste 20 de ani au impus înlocuirea DES-ului care a devenit vulnerabil. S-a demonstrat de curând că, folosind o mașină paralelă complexă ce costă 250.000 USD, se poate găsi, într-un timp de aproximativ 60 de ore, o cheie de 56 de biți cu care a fost criptat un bloc de text clar. Ca urmare, s-a lansat anul trecut un concurs de cifruri pentru realizarea așa numitului *AES* (*Advanced Encryption Standard*). Până la ora actuală există deja 16 propuneri de algoritmi criptografici sime-

trici pentru AES. Pentru a înțelege mai bine cum este "în interior" un sistem de criptare, prezentăm ***cifrul FROG***, realizat de o echipă de la *TecApro International*. Acesta, ca orice alt cifru simetric își propune să transforme textul clar în text cifrat, printr-un proces de calcul care să creeze *confuzie* și *difuzie*.

FROG criptează blocuri de intrare de 16 octeți în blocuri de ieșire cu aceeași lungime, sub controlul unei *chei criptografice utilizator*, a cărei lungime poate varia între 5 și 125 de octeți.

Cifrul realizează **8 iterații (runde)** în prelucrarea fiecărui bloc de intrare. Fiecare iterație folosește o *cheie internă*, generată din cheia de cifrare și care este o structură de date cu 8 înregistrări, fiecare având 3 câmpuri:

- *xorBu* - un vector de 16 octeți folosit în *sau exclusiv*;
- *substPermu* - un vector de 256 octeți folosit ca tabelă de substituții;
- *bombPermu* - un vector de 16 octeți folosit în a referi diferite poziții în bloc (cu valori între 0 și 15), pentru a realiza o permutare.

Fiecare iterație traversează secvențial blocul de 16 octeți, de la stânga la dreapta și calculează cu fiecare octet ***4 operații de bază***: 2 implementează ***confuzie***, celelalte 2 ***difuzie***:

- **Pasul 1**: SAU-EXCLUSIV(XOR) între octetul curent din bloc și octetul curent din câmpul *xorBu*.
- **Pasul 2**: Înlocuirea octetului calculat la pasul 1 prin octetul din tabela de substituție *substPermu*, indexată cu octetul calculat la pasul 1.
- **Pasul 3**: Modificarea octetului următor din bloc prin XOR cu octetul calculat la pasul 2. Când se ajunge la sfârșitul blocului, octetul următor este considerat primul.
- **Pasul 4**: Se folosește octetul curent al lui *bombPermu* pentru a defini o poziție în bloc. Se modifică octetul din acea poziție făcând XOR cu octetul calculat în pasul 2.

Câmpul *bombPermu* este o permutare, a cărei lungime a ciclului trebuie să fie egală cu dimensiunea blocului (16).

Un ***exemplu numeric*** va clarifica modul de lucru al cifrului. Toate valorile sunt date în hexazecimal, începând cu cel mai semnificativ octet. Vom defini o cheie internă cu următoarele valori:

```
xorBu: 05 f0 a3 ...
substPermu: a2 16 08 bb 03 f1 ...
bombPermu: 03 0f 00 ...
```

Să realizăm cifrarea următorului bloc de 16 octeți de text clar:

`00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0f`

și să arătăm stările intermediare ale blocului după fiecare pas.

Prelucrarea octetului din poziția 0:

Pasul 1:
`05 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0f`
Pasul 2:
`f1 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0f`
Pasul 3:
`f1 f0 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0f`
Pasul 4:
`f1 f0 02 f2 04 05 06 07 08 09 0a 0b 0c 0d 0f`

Prelucrarea octetului din poziția 1:
Pasul 1:
`f1 00 02 f2 04 05 06 07 08 09 0a 0b 0c 0d 0f`
Pasul 2:
`f1 a2 02 f2 04 05 06 07 08 09 0a 0b 0c 0d 0f`
Pasul 3:
`f1 a2 a0 f2 04 05 06 07 08 09 0a 0b 0c 0d 0f`
Pasul 4:
`f1 a2 a0 f2 04 05 06 07 08 09 0a 0b 0c 0d ad`

Prelucrarea octetului din poziția 2:
Pasul 1:
`f1 a2 03 f2 04 05 06 07 08 09 0a 0b 0c 0d ad`
Pasul 2:
`f1 a2 bb f2 04 05 06 07 08 09 0a 0b 0c 0d ad`
Pasul 3:
`f1 a2 bb 49 04 05 06 07 08 09 0a 0b 0c 0d ad`
Pasul 4:
`4a a2 bb 49 04 05 06 07 08 09 0a 0b 0c 0d ad`

Procesul continuă în aceeași manieră pentru toți cei 16 octeți. Procesul se repetă cu alte chei interne de 8 ori, adică se execută 8 runde identice.

Decriptarea se face parcurgând cheia internă și blocul cifrat, în sens invers. Cheia internă folosită este aceeași, cu excepția tabelelor de substituție - *substPermu* - care sunt înlocuite de inversele lor.

Un lucru important în evoluția algoritmului îl reprezintă ***inițializarea cheii interne***. **FROG** calculează cheia internă pe baza cheii criptografice furnizate de user. Fiecare iterație folosește un articol de cheie internă cu 16+256+16 = 288 octeți. Ca urmare, cheia internă totală are 288 * 8 = 2304 octeți. Pentru generarea ei se folosesc doi algoritmi:

1. Primul algoritm (funcția *hashKey*) folosește cheia user (*simpleKey*) și o constantă internă (*randomSeed*) și produce un tablou de 2304 octeți (numit *randomKey*), printr-un algoritm complex în 5 pași.
2. Al doilea algoritm (funcția *makeInternalKey)* ia tabloul arbitrar de 2304 octeți, generat de primul algoritm, și îl transformă într-o structură ce cuprinde cele 3 câmpuri. Se calculează corect vectorii de permutare *substPermu* și *bombPermu* pentru toate cele 8 înregistrări.

Cel mai eficient atac la un astfel de cifru este acela ce încearcă deducerea cheii criptografice prin căutare exhaus-

tivă. Factorul de muncă necesar este foarte mare (prohibitiv): $O(2(N-1))$, unde N reprezintă numărul de biți ai cheii utilizator. De exemplu $O(2\ (255))$ pentru o cheie de 32 de caractere (256 biți).

Moduri practice de cifrare a datelor

În practica criptării, apare necesitatea criptării unor informații de lungime oarecare (fișiere, mesaje, e-mail, parole etc.). În acest scop, există *două moduri principale de utilizare a cifrurilor*:

- ***Cifrarea bloc*** operează cu blocuri de text clar și cifrat - de regulă de 64 de biți, dar uneori și mai mari. În acest caz, același bloc de text clar va fi cifrat de fiecare dată în același bloc de text cifrat, folosind aceeași cheie.
- ***Cifrarea secvențială*** operează cu secvențe de text clar și cifrat de un bit sau octet (uneori chiar și cu cuvinte de 32 de biți). În cifrarea secvențială, secvențe similare de text clar vor fi cifrate diferit, în cazul unor cifrări repetate.

Modurile de criptare practică constituie combinații ale celor două tipuri de bază. Vom exemplifica câteva scheme foarte răspândite.

Cifrare carte de coduri (Electronic Codebook-ECB)

ECB este cea mai obișnuită cale de cifrare bloc, prin care un bloc de text clar se transformă într-un bloc de text cifrat. De vreme ce același bloc de text clar se cifrează întotdeauna în același bloc de text cifrat, teoretic este posibilă crearea unei cărți de coduri în care să se facă asocierea *text clar - text cifrat*. Pentru blocuri de 64 de biți rezultă un număr de 264 de intrări în cartea de coduri - mărime mult prea mare pentru a permite memorarea și manipularea lor. În plus, fiecare cheie necesită propria carte de coduri. Dacă notăm cu C_i blocurile de criptogramă, cu P_i blocurile de text clar și cu K cheia, avem:

la criptare: $C_i = E_k\ (P_i)$
la decriptare: $P_i = D_k\ (C_i)$

S-a notat cu E_k *funcția de cifrare* (de exemplu FROG sau DES sau IDEA) cu cheia K și cu D_k *funcția de decriptare* cu aceiași cheie. Acesta este cel mai simplu mod de lucru, în care fiecare bloc de text clar este cifrat independent. Fișierul ce se criptează nu trebuie să intre la cifrare liniar, de la început la sfârșit. Criptarea se poate face luând aleator blocuri din cadrul fișierului. Acest lucru este important pentru fișierele criptate care sunt accesate aleator, ca de exemplu în cazul bazelor de date. Dacă o bază de date este criptată cu *ECB* atunci orice înregistrare poate fi adăugată, ștearsă, criptată sau decriptată independent de celelalte - considerând că înregistrările conțin un număr întreg de blocuri criptate.

Problema *ECB*-ului poate fi formulată astfel: un criptanalist, care deține textul clar și textul cifrat echivalent pentru câteva mesaje, poate realiza o carte de coduri fără a cunoaște cheia? În exprimarea curentă sunt fragmente de mesaje care tind să se repete. Mesajele de comunicație între calculatoare, cum ar fi poșta electronică, au structuri regulate. Mesajele pot avea structuri redundante sau șiruri lungi de spații sau zerouri. Dacă criptanalistul realizează că mesajul în clar `5e081bc5` se criptează în textul cifrat `7ea593a4`, el poate decripta imediat respectivul mesaj acolo unde îl întâlnește. Mai mult, dacă mesajele criptate au un număr mare de redundanțe și acestea tind să apară în aceleași locuri în mesaje diferite, atunci un criptanalist poate estima o serie de informații. El poate intensifica atacul statistic asupra textului de bază, indiferent de tăria cifrului bloc.

Cele mai multe mesaje nu se împart exact în blocuri de 64 de biți (sau orice altă mărime), de regulă rămânând un bloc scurt la sfârșitul mesajului. *ECB* necesită blocuri de 64 de biți. Această problemă se reolvă prin *Completarea blocurilor* (*padding*). Se completează ultimul bloc cu un model zero-unu, alternând cifrele de 1 și 0, astfel încât blocul să devină complet. Deoarece este nevoie la decriptare de ștergerea acestor surplusuri, ultimul octet al ultimului bloc va conține numărul de octeți adăugați. De exemplu, un mesaj care după împărțirea în blocuri de 64 de biți rămâne cu ultimul bloc de doar 3 octeți, se completează cu 4 octeți, iar ultimul octet va conține numărul 5. La decriptare, se citește ultimul octet și se șterg ultimii 5 octeți.

Cifrarea bloc cu înlănțuire (Cipher Block Chaining-CBC)

Acest mod adaugă mecanismului de criptare un bloc de *reacție*. Rezultatul criptării unui bloc anterior revine prin buclă și intervine în criptarea blocului curent. În felul acesta textul cifrat nu mai depinde doar de textul clar, ci și de cifrarea blocului anterior.

În *CBC*, textul clar, înainte de a intra în blocul de criptare propriu-zis, este însumat modulo 2 (XOR) cu blocul de text cifrat anterior. Figura 1 prezintă modul de criptare CBC.

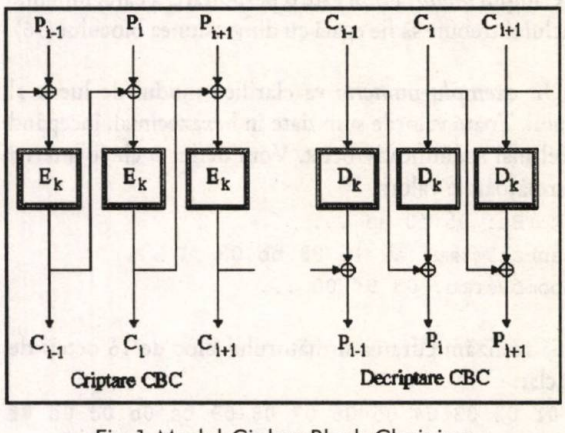


Fig.1 Modul Cipher Block Chaining

Procesul se poate nota astfel:
la criptare: $C_i = E_k\ (P_i \oplus C_{i-1})$
la decriptare: $P_i = C_{i-1} \oplus D_k(C_{i-1})$

CBC transformă același text în blocuri de text cifrat, diferite. Două mesaje identice se vor transforma practic în același mesaj criptat. Mai mult, două mesaje care încep la fel, vor fi criptate identic până la prima diferență.

Prevenirea acestui lucru se poate face criptând primul bloc cu un vector de date aleator. Acest bloc de date aleatoare se numește *vector de inițializare* al schemei, *variabilă de inițializare* sau *valoare inițială pentru înlănțuire.* Vectorul de inițializare nu are nici un înțeles de sine stătător, locul lui este dat de faptul că poate produce criptarea unică a oricărui mesaj, chiar și a mesajelor identice. Când receptorul decriptează acest bloc, el îl utilizează pentru inițializarea registrului buclei de reacție.

O marcă de timp constituie un bun exemplu de vector de inițializare. Prin adăugarea vectorului de inițializare, mesaje de text clar identice vor fi criptate diferit. Astfel, pentru o persoană indiscretă este imposibil să facă un *bloc replay* și mult mai dificil să realizeze o carte de coduri. Vectorul de inițializare nu trebuie să fie secret; el poate fi transmis în clar împreună cu mesajul cifrat.

Cifrarea secvențială (Stream Ciphers)

Prin acest mod, textul clar se convertește octet cu octet în text cifrat. Modelul general este dat în figura 2. Un generator de chei generează un șir de octeți K_1, K_2, ..., K_i. Cu acest șir cheie se face XOR (sumă modulo 2) cu șirul de octeți ai textului clar, P_1,P_2,...,P_i, pentru a produce un șir de text criptat:

la criptare: $C_i = P_i \oplus K_i$

La decriptare, rezultatul este obținut din efectuarea unui XOR între criptogramă și aceeași cheie curentă produsă de același generator:

la decriptare: $P_i = C_i \oplus K_i$

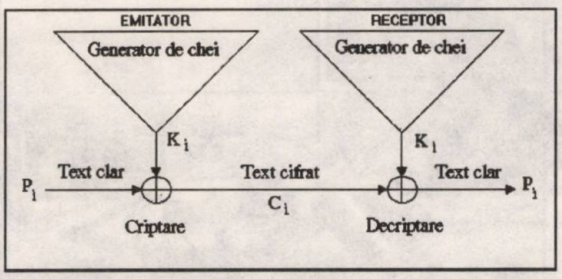


Fig.2 Modul de criptare secvențială

Securitatea sistemului depinde în întregime de generatorul de chei. În practică se folosește o funcție de criptare simetrică (*FROG* sau *DES*) drept generator de cheie.

Criptarea triplă cu două chei (Triple Encryption)

O idee mai "tare", prelucrează un bloc de trei ori, folosind doar două chei. Se recomandă ca expeditorul mesajului să cripteze prima dată cu prima cheie, apoi să decripteze cu cea de-a doua și în final să cripteze din nou cu prima cheie. Algoritmul poate fi oricare dintre cele simetrice (DES, IDEA, FROG etc.). La decriptare, destinatarul urmează procesul invers: decriptare cu prima cheie, criptare cu cea de-a doua, decriptare cu prima cheie:

$$C = E_{k1}(D_{k2}(E_{k1}(P)))$$
$$P = D_{k1}(E_{k2}(D_{k1}(C)))$$

Această metodă se mai numește și *EDE (Encrypt-Decrypt-Encrypt)*. Dacă algoritmul utilizează o cheie de *n* biți, atunci schema va folosi o cheie de *2n* biți. Modelul criptare-decriptare a fost adăugat la standardul *DES* în X9.17 și în standardele ISO 8732.

Utilizarea unui ***algoritm de compresie*** împreună cu criptarea are sens din două motive:

- criptarea este mare consumatoare de timp; compresia unui fișier anterior operației de criptare mărește simțitor viteza procesului;
- în "spargerea" unui cifru se pot folosi redundanțele din textul clar; la compresie numărul acestora se reduce.

Este important să se efectueze compresia înainte de criptare, deoarece dacă algoritmul de cifrare este bun, textul cifrat nu va fi compresibil, el fiind un un șir de date aleatoare.

Cum alegem criptarea în aplicațiile din rețele

Putem privi securitatea (datelor, comunicațiilor, în general a oricărui tip de informații) ca un lanț. Securitatea întregului sistem este o combinație puternică de legături slabe.

Totul trebuie securizat: algoritmii criptografici, protocoalele, programele de administrare etc. Criptografia este doar o parte a securității; ea acoperă problematica realizării securității unui sistem, ceea ce este diferit de ceea ce înseamnă realizarea unui sistem securizat. Tradiționala imagine a criptografiei, ca spion în tehnologie, este destul de departe de realitate.

Peste 99% din aplicațiile criptografice utilizate în lume nu protejează secrete militare; ele sunt întâlnite în bănci, plăți-TV, taxe de drum, acces la terminale, contoare de electricitate, carduri etc.

Rolul criptografiei în aceste aplicații este de a împiedica furturile și înșelăciunile. În cele mai multe dintre aceste aplicații nu s-a utilizat eficient criptografia, atacurile reușite neavând nimic în comun cu criptanaliza. Chiar și *NSA* (*National Security Agency*) a admis că marea parte a erorilor, apărute în activitățile sale, provin din erorile de implementare și nu din algoritmi sau protocoale. În aceste condiții, nu contează cât de bună a fost criptografia, atacurile reușite anulând acest lucru și speculând erorile de implementare.

Pentru ***alegerea și evaluarea unui algoritm***, există o serie de variante:

- se poate alege un algoritm publicat, în ideea că un astfel de algoritm a fost analizat de mulți criptanaliști și, dacă încă nu a fost spart, el este bun;
- se poate utiliza un produs al unui fabricant, acesta din urmă având interesul de a realiza produse cât mai performante;
- se poate apela la serviciile unui specialist în domeniu;
- se poate scrie un algoritm propriu.

Fiecare dintre aceste alternative este problematică, însă prima este cea mai pretabilă, restul putând genera probleme. Mulți dintre cei care fac apel la consultanți nu au de obicei noțiuni despre criptare. Pentru *NSA* lucrează câțiva dintre cei mai buni criptanaliști, dar aceștia cu siguranță nu vor spune tot ceea ce știu. Chiar dacă sunteți un geniu, a scrie un algoritm și a-l utiliza fără a afla opiniile altora, constituie o greșeală.

Algoritmii prezentați în diferite lucrări sunt publici. Cei mai mulți au apărut în literatura de specialitate și majoritatea au fost criptanalizați de experți în domeniu. Literatura de specialitate prezintă rezultatele testărilor și studiilor, atât pe cele pozitive cât și pe cele negative. ***Atenție însă la algoritmii de export!*** Algoritmii pentru export în afara S.U.A. trebuie aprobați de guvern (concret, de către *NSA*). Este destul de răspândită opinia că algoritmii autorizați de către *NSA* pot fi sparți de către aceasta. Există câteva lucruri pe care *NSA* evită să le spună firmelor străine cărora le aprobă folosirea produselor criptografice:

- punerea câte unui bit de cheie, la un moment dat, în textul cifrat;
- scurtarea cheii la o dimensiune mult mai mică; de exemplu, dacă algoritmul admite o cheie de 8 octeți, din cheie să se folosească numai 4 octeți;
- criptarea unui header fixat sau a unui șir aleator la începutul fiecărui mesaj criptat, facilitându-se atacul prin text clar cunoscut.

NSA păstrează o copie a codului sursă al programelor criptografice aprobate pentru export, însă detaliile algoritmului rămân secrete. Cu certitudine, nimeni nu a făcut publică vreuna dintre aceste slăbiciuni, însă trebuie să se știe aceasta când se cumpără un produs criptografic din S.U.A. care a fost aprobat pentru export.

În numărul viitor vom vedea cum lucrează noile sisteme de criptare cu cheie publică și cum sunt ele folosite în Internet.

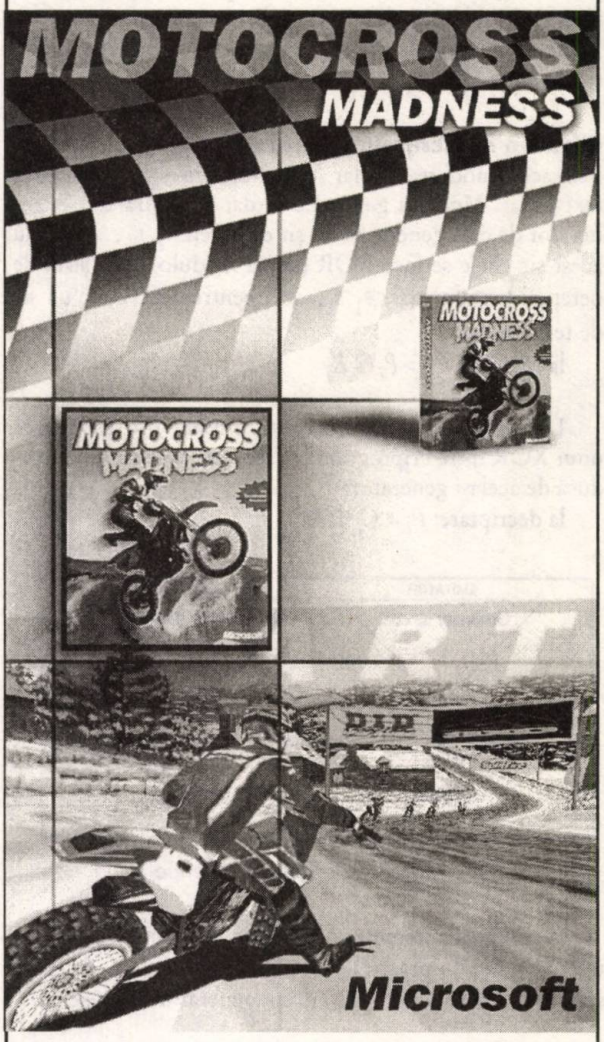
Domnul prof. dr. Victor-Valeriu Patriciu este cadru didactic la Academia Tehnică Militară. Poate fi contact prin e-mail la adresa: vip@mta.ro.

Premiul lunii

Și în acest număr, revista noastră oferă un *PREMIU DE FIDELITATE* unui cititor ales prin tragere la sorți dintre *ABONAȚI*.

Și tu ai șansa să câștigi *PREMIUL LUNII*. Condiția este simplă: *ABONEAZĂ-TE !*

Premiul lunii martie este oferit de sponsorul nostru, *MICROSOFT ROMANIA*...



... și a fost câștigat de către

Bogdan Cosmin Bucur **din București.**

Descoperiți
modul de lucru Visual

Acum vă puteți permite(*)
un instrument de lucru original

(*) peste 70% reducere
numai pentru persoane fizice
numai pentru uz personal, acasă

Oferta este valabilă în perioada 01.04 - 30.06 1999

Denumirea produsului	Preț (USD)
VB Std 6.0 Win 32 English Intl CD Lmg	31
VB Pro 6.0 Win 32 Romanian CVUP CD	71
VB Pro 6.0 Win 32 Romanian CD	101
VC++ Std 6.0 Win 32 English Intl CD	31
VC++ Pro 6.0 Win 32 English Intl CVUP CD	71
VC++ Pro 6.0 Win 32 English Intl CD	101

Denumirea produsului	Preț (USD)
VFoxPro Pro 6.0 Win 32 English Intl CVUP CD	71
VFoxPro Pro 6.0 Win 32 English Intl CD	101
VInterDEV 6.0 Win 32 English Intl CVUP CD Pro	71
VInterDEV 6.0 Win 32 English Intl CD Pro	101
VJ++ Std 6.0 Win 32 English Intl CD	31
VJ++ Pro 6.0 Win 32 English Intl CVUP CD	71
VJ++ Pro 6.0 Win 32 English Intl CD	101

Prețul include 70% discount și se plătește în lei, la cursul de schimb al B.N.R. Prețul nu include T.V.A.

Pentru a beneficia de această ofertă,
contactați firma GeCAD, Microsoft Developer Dealer, tel.: 01-324 84 09,
http://www.gecad.ro/mspromo

PERSPECTIVA 2000

Microsoft®

FIRE de executare*)

Horia Georgescu

Programarea obișnuită presupune existența unui "dispozitiv" de executare a instrucțiunilor; instrucțiunile unicului program se execută secvențial, pas cu pas, de către unicul procesor existent.
În viața reală asistăm în permanență la mai multe acțiuni ce se desfășoară simultan. Unele dintre ele sunt independente, altele au un grad mai mic sau mai mare de dependență: încep în același timp, se așteaptă unele pe altele, sunt serializate etc.

Calculatoarele moderne realizează, atât prin resurse software cât și hardware, desfășurarea concomitentă a mai multor acțiuni. Posibilitatea de a lansa și gestiona mai multe *fire de executare* este principala noțiune nou introdusă de sistemul de operare Windows 95.

Acțiunile ce se desfășoară concomitent pot fi foarte variate: programe "tradiționale", secvențe de animație, sunet etc. În acest capitol vom presupune că *acțiunile* sunt *secvențe de instrucțiuni.*

Pentru prelucrarea separată a fiecărei acțiuni, va fi *lansat* un fir de executare, care va prelua sarcina de a executa instrucțiunile respective. Interpretorul Java este capabil (de fapt se lucrează în prezent la această facilitate) să detecteze dacă mașina gazdă dispune de mai multe procesoare, caz în care acestea sunt efectiv folosite pentru executarea concomitentă a mai multor acțiuni.

În cazul existenței unui singur procesor sau mai general, atunci când numărul de acțiuni ce trebuie executate depășește numărul de procesoare, executarea concurentă este simulată: unitatea centrală alocă pentru fiecare fir de executare cuante de timp.

Elementele legate de programarea firelor de executare fac parte integrantă din limbajul Java, în acest mod compilatorul Java putând "vedea" aceste fire. În acest mod este justificată afirmația că *Java este un mediu de executare.*

Crearea firelor de executare

O primă modalitate de a crea și lansa în executare fire (de executare) este de a folosi clasa `Thread` din pachetul **`java.lang`**.

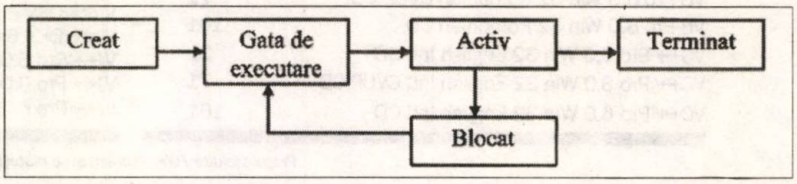
Crearea unui fir de executare se începe prin crearea unui obiect de tipul `Thread`:

```
Thread fir = new Thread();
```

După ce obiectul a fost creat, el poate fi configurat și apoi se poate trece la executarea sa. Configurarea cuprinde elemente ca, de exemplu, precizarea unui nume, a unei priorități etc. Când firul este gata de executare, se invocă metoda sa `start` fără argumente. Ca urmare, firul de executare devine activ; mai precis, este executată metoda sa `run`.

Când metoda `run` se încheie, firul de executare se termină. Terminarea poate fi realizată și explicit prin invocarea metodei `stop`. Menționăm că executarea mai poate fi suspendată cu metoda `suspend` sau manevrată în alte moduri, dintre care unele vor fi descrise în continuare.

Stările în care se poate afla un fir de executare sunt următoarele:



Un fir (de executare) este *gata de executare* dacă îndeplinește toate condițiile pentru a se trece la executarea sa, dar nu i s-a alocat încă un procesor; când un procesor liber preia firul, acesta devine *activ*. *Blocarea* unui fir de executare poate fi realizată pe baza unei condiții (de exemplu un semafor).

Implementarea standard a metodei `Thread.run` nu prevede vreo acțiune. De aceea este necesar ca programatorul să

*) În *"Mic dicționar enciclopedic"*, editura *Științifică și Enciclopedică*, București, 1978, în dreptul cuvântului ***execuție*** găsim explicația: *aducerea la îndeplinire a unei hotărâri de condamnare la moarte*, iar în dreptul cuvântului ***executare*** avem: *acțiunea de a executa și rezultatul ei; îndeplinire.*

extindă clasa Thread și să rescrie (suprapună) metoda run.

Se impun câteva precizări privind firele de executare.

Firul principal de executare folosește ca punct de plecare programul respectiv, cu precizarea clasei principale (ce are același nume ca fișierul corespunzător) și știe că trebuie să execute instrucțiunile din metoda "principală" main.

Un fir de executare este lansat de un obiect ce instanțiază o clasă ce extinde clasa Thread. Fie Tip numele clasei și Ob numele obiectului. Lansarea se face prin apelul Ob.start().

Firul folosește (ca și firul principal) programul curent, cu deosebirea că acum "clasa principală" este considerată a fi instanțierea clasei Tip reprezentată prin obiectul Ob. În plus, firul de executare știe că trebuie să execute instrucțiunile din metoda "principală" run a clasei Tip.

Să presupunem că în clasa Tip mai apare o metodă met. Trebuie clarificată diferența între apelul Ob.start() și apelul Ob.met(...). Apelul Ob.met(...) are următorul efect: obiectul Ob este "pus" să execute metoda met și nu se trece mai departe decât după ce această activitate a fost încheiată. În schimb, apelul Ob.start() realizează lansarea unui fir de executare pe baza obiectului Ob; deci obiectul Ob este folosit ca model de plecare pentru noul fir de executare și nu este "pus" să execute ceva. În continuare noul fir de executare își desfășoară activitatea concurent cu firul care l-a lansat. Obiectul Ob nu este "ocupat" cu vreo activitate, deci în continuare el poate fi "pus" să execute acțiuni. De exemplu, dacă apoi obiectul este pus să execute o metodă sau să modifice un câmp al clasei a cărei instanțiere este, el poate influența activitatea firului de executare lansat pe baza lui; această posibilitate este folosită frecvent pentru a controla terminarea firelor de executare.

Exemplu:

Să considerăm următorul program:

```
class Tip extends Thread {
  int k=0; boolean continua=true;
  public void run() {
    while (continua)
      System.out.print(" "+k++);
    System.out.println("Fir"); stop();
  }
  public void terminare() {
    continua=false;
  }
}
class P1 {
  public static void main(String[] w)
  throws java.io.IOException {
    Tip Ob = new Tip(); Ob.start();
    while ((char)System.in.read() != '\n');
      Ob.terminare();
    System.out.println("main");
  }
}
```

Pe baza obiectului Ob este lansat un fir de executare. Activitatea acestuia constă în a tipări, atâta timp cât variabila booleană continua are valoarea true, șirul numerelor naturale. Firul de executare principal prevede citirea repetată de caractere, până când este introdus <Enter>; în acel moment obiectul Ob execută metoda terminare, al cărei efect este modificarea valorii lui continua în false, ceea ce va determina oprirea firului de executare lansat plecând de la obiectul Ob. Firul de executare principal se termină și el. Să remarcăm că nu este previzibil dacă mesajul "Fir" va fi tipărit înainte sau după mesajul "main".

Un prim mod de sincronizare

Vom prezenta un prim mod de sincronizare, cu largă aplicabilitate în transcrierea în Java a algoritmilor paraleli.

Să ne situăm în cazul în care un fir de executare lansează unul sau mai multe alte fire de executare și, prin natura problemei, trebuie să aștepte terminarea activităților acestora înainte de a întreprinde o nouă acțiune (în lipsa unei astfel de măsuri, firul principal își continuă activitatea în paralel cu firele pe care le-a lansat).

Pentru rezolvarea problemei poate fi folosită metoda join a clasei Thread. Ea are următoarele două forme:

- `public final void join() throws InterruptedException`
- `public final synchronized void join(long mili) throws InterruptedException`

Prima formă face ca firul de executare care a lansat la rândul său un fir de executare să aștepte terminarea acestuia un timp nelimitat, înainte de a trece la următoarea acțiune prevăzută în program. A doua formă limitează așteptarea la un număr specificat de milisecunde.

Exemplu:

Să considerăm următorul program:

```
import java.util.*;
class Tip extends Thread {
  static int k=1;
  public void run() {
    for (int i=0; i<100; i++)
      System.out.print(k+" ");
    stop();
  }
}
class P2 {
  public static void main(String[] w) {
  Tip Ob = new Tip(); Ob.start();
  try {Thread.sleep(100);}
  catch (InterruptedException e) { }
  Ob.k=2;
  try { Ob.join(); }
  catch(InterruptedException e) { }
  System.out.println("***");
  }
}
```

După ce lansează un nou fir de executare pe baza obiectului `Ob`, firul principal își întrerupe activitatea 100 milisecunde, apoi modifică în 2 valoarea câmpului `k` al clasei `Tip` și așteaptă ca acesta să își încheie activitatea, înainte de a tipări 3 asteriscuri și de a-și termina la rândul său executarea. Drept urmare, la ieșire va apare un șir de `1`, urmat de un șir de `2`.

Firul principal de executare nu este cu nimic deosebit de cele pe care le lansează; în particular, aceasta înseamnă că firul principal, după ce lansează un fir, își poate încheia activitatea, întreaga aplicație încheindu-se atunci când firul nou lansat se termină. Afirmația de mai sus este valabilă pentru orice fir (nu neapărat cel principal) ce lansează alte fire.

Este recomandabil ca un fir de executare care lansează alte fire să nu se termine înaintea acestora. Această recomandare face parte din disciplina de programare în Java.

Despre programarea paralelă

Numeroase aplicații științifice necesită în zilele noastre calculatoare cu putere de calcul ce depășește cu mult un miliard de instrucțiuni pe secundă. Simularea unor fenomene fizice (de exemplu previziunile meteorologice, predicțiile mareelor sau comportările aerodinamice) este limitată atât din punct de vedere al soluțiilor cât și al preciziei acestora de resursele de care se dispune din punct de vedere informatic.

Performanța unui calculator se poate exprima fie prin numărul operațiilor în virgulă mobilă pe secundă (*Mflops*), fie prin numărul de instrucțiunilor pe care este capabil să le execute pe secundă (*Mips*). Încercarea de creștere a acestor performanțe, prin proiectarea unor circuite cât mai sofisticate, s-a izbit de anumite limitări fizice: disiparea unei cantități mari de energie într-un volum redus, probleme legate de accesul la memorie etc., cât și de costul mult prea ridicat al unor astfel de echipamente.

O soluție alternativă, radical diferită, a fost utilizarea unor arhitecturi paralele. Ideea de bază de la care s-a pornit a fost introducerea unui număr mare de procesoare având o putere și un cost relativ scăzut, care cooperează la rezolvarea unei probleme ce necesită un calculator cu o putere mare de calcul.

În dezvoltarea acestor arhitecturi, interesul s-a axat nu numai pe performanțele maxime ce pot fi atinse, ci și pe adaptabilitatea acestor echipamente la cerințele diverselor aplicații.

Există astăzi un număr mare de calculatoare paralele și o diversitate de arhitecturi, unele fiind încă într-un stadiu experimental sau de prototip. Aceste arhitecturi se disting mai ales prin numărul și complexitatea procesoarelor utilizate: spectrul lor variază de la un număr scăzut de procesoare de putere ridicată, cum ar fi CRAY 2, la mii de procesoare simple cum ar fi Connection Machine.

În momentul în care se pune problema proiectării unui sistem paralel trebuie ținut cont de o serie de aspecte: nivelul la care se realizează paralelismul, modul de comunicare între procesoare, topologia lor (modalitatea de interconectare). Fiecare din aceste aspecte poate constitui un criteriu de clasificare.

În cele ce urmează vom preciza modelul de calcul paralel cu care vom lucra.

Un prim aspect este cel legat de *modul în care procesoarele comunică între ele*. În continuare vom folosi modelul ***P-RAM*** (*Parallel Random Access Machine*), care presupune că toate procesoarele au acces la o memorie comună. În acest mod nu se impun restricții ce țin de resursele fizice disponibile (hardware). Se presupune că fiecare procesor comunică direct în ambele sensuri cu memoria comună; dacă procesorul P_i dorește să transmită o valoare procesorului P_j, procesorul P_i va scrie valoarea într-o locație de memorie, iar P_j va citi valoarea din acea locație de memorie.

P-RAM este un model cu *memorie distribuită:* un număr de procesoare lucrează sincron și comunică prin intermediul unei memorii comune dinamice (*random access machine*). Fiecare procesor poate fi accesat în mod aleator (*uniform-cost random access machine*) și poate efectua operații și instrucțiuni uzuale.

Un al doilea aspect este legat de *activitatea procesoarelor*. Vom alege modelul ***SIMD*** (*Single Instruction, Multiple Data Stream*) care presupune că toate procesoarele efectuează aceleași operații, dar pe seturi de date diferite. Menționăm doar că un model destul de folosit este și modelul ***MIMD*** (*Multiple Instruction, Multiple Data Stream*).

Un al treilea aspect se referă la drepturile pe care le au mai multe procesoare *la scrierea și citirea concomitentă* (paralelă) în/dintr-o locație de memorie. Mai precis, trebuie specificat dacă mai multe procesoare pot sau nu citi concomitent din aceeași locație de memorie, respectiv dacă mai multe procesoare pot scrie concomitent în aceeași locație de memorie.

Vom presupune că folosim modelul ***CREW*** (*Concurrent Read, Exclusive Write*), care permite citirea concomitentă din aceeași locație de memorie, dar interzice scrierea concomitentă (concurentă) în aceeași locație de memorie.

Forma generală a unei instrucțiuni paralele este:

```
for toți x∈X in parallel
        instrucțiune(x)
```

unde `X` este o mulțime finită.

Semnificația acestei instrucțiuni este următoarea:
- fiecărui element `x` din mulțimea `X` i se atribuie câte un procesor;
- fiecare dintre aceste procesoare execută în paralel cu celelalte instrucțiunea menționată, bineînțeles pentru elementul `x∈X` respectiv.

Executarea instrucțiunii paralele se termină când toate procesoarele și-au terminat propriile calcule. Această sin-

cronizare este cea tipică programării paralele. Ea poate fi rezolvată folosind elementele din paragraful precedent. În paragraful următor vom exemplifica modul de implementare al instrucțiunii paralele pe o metodă generală de programare din calculul paralel.

Prezentăm în continuare modul în care sunt atribuite și activate procesoarele la executarea instrucțiunii paralele.

Atribuirea (asignarea) *procesoarelor* se realizează astfel: elementele lui X sunt (sau pot fi codificate ca) numere întregi. Atunci fiecărui element $x \in X$ îi vom asocia procesorul $P_{cod(x)}$, unde $cod(x) \in \{1,\ldots,n\}$ este codul corespunzător lui x; n este cardinalul mulțimii X. Se presupune că fiecare procesor P_i va fi "înștiințat" că trebuie să lucreze pentru $x=cod^{-1}(i)$.

Ideea conform căreia un algoritm ce rulează pe p procesoare va determina soluția de p ori mai repede, este greșită. În primul rând este bine de știut că există probleme care sunt greu sau chiar imposibil de paralelizat, iar în interiorul unui proces vor exista instrucțiuni care se vor executa secvențial.

Timpul de calcul paralel este perioada care s-a scurs de la inițierea primului proces și momentul când toate procesele au fost terminate. El depinde nu numai de complexitatea operațiilor, ci și de complexitatea comunicării și de sincronizare, respectiv de limitele schimbului de date. Performanțele unui program sunt desigur influențate și de numărul procesoarelor utilizate.

Implementarea metodei arborelui binar în Java

Este vorba de o metodă generală, cu largă aplicabilitate, din programarea paralelă.

Numele ei provine de la faptul că este utilizat un *arbore binar complet*. Reamintim că un arbore binar complet este un arbore binar cu 2^k-1 vârfuri în care:

- vârfurile sunt situate pe nivelurile $0,1,\ldots,k-1$;
- vârfurile situate pe nivelurile $0,1,\ldots,k-2$ au exact doi descendenți, iar cele de pe nivelul $k-1$ sunt vârfuri terminale (frunze);
- vârfurile de pe un nivel i oarecare sunt numerotate cu $2^i..2^{i+1}-1$; descendenții vârfului i sunt vârfurile 2i și 2i+1.

Fie vectorul $a=(a_1,\ldots,a_n)$ și o operație asociativă $\otimes$. Se urmărește calculul valorii $a_1 \otimes a_2 \otimes \ldots \otimes a_n$.

Vom presupune că n este o putere a lui 2 ($n=2^m$). În caz contrar vom completa vectorul la dreapta, până se obține un vector a cărui lungime este o putere a lui 2, cu elementul neutru al operației $\otimes$:

- 0 dacă $\otimes$ este operatorul de sumare;
- 1 dacă $\otimes$ este operatorul de înmulțire;
- $+\infty$ dacă $\otimes$ este operatorul de minim;
- $-\infty$ dacă $\otimes$ este operatorul de maxim;
- 0 dacă $\otimes$ este operatorul de disjuncție logică;
- 1 dacă $\otimes$ este operatorul de conjuncție logică.

Vom construi un arbore binar complet și vom atașa valori vârfurilor sale. Rădăcina corespunde calculului valorii finale cerută de problemă. Fiecare nod intern corespunde unei valori parțiale; la același nivel (adâncime în arbore) calculele se execută în paralel. Metoda este de tip bottom-up, adică vom parcurge arborele pe niveluri, plecând de la frunze și mergând spre rădăcină.

Într-o primă etapă dublăm dimensiunea vectorului și mutăm elementele date inițial pe pozițiile $a_n,\ldots,a_{2n-1}$:

```
for i=1,n in parallel do
  a_{n-1+i} ← a_i                                {1}
```

În a doua etapă parcurgem arborele de jos în sus, atribuind valori vârfurilor interne. Valoarea fiecărui vârf se calculează în funcție de valorile descendenților săi, folosind operatorul $\otimes$.

```
for k=m-1,0,-1 do
  for i=2^k,2^{k+1}-1 in parallel                {2}
    a_i ← a_{2i} ⊗ a_{2i+1}
```

Să observăm că valoarea a_i atașată unui vârf i este "suma" valorilor atașate frunzelor din subarborele de rădăcină i. Prin urmare rezultatul se obține în a_1.

Se folosesc n *procesoare. Timpul* este $O(m)=O(\log n)$, deoarece ciclul **for** interior necesită timp constant și se execută de m ori.

Ilustrăm cele de mai sus pentru *calculul sumei a* n *numere.*

Considerăm vectorul a=(3,7,8,3,9,2,3,1) cu $n=8=2^3$ și m=3. Operatorul $\otimes$ va fi în acest caz operatorul de sumare. Figura următoare ilustrează fiecare pas al algoritmului.

Programul de mai jos transcrie în limbajul Java metoda arborelui binar.

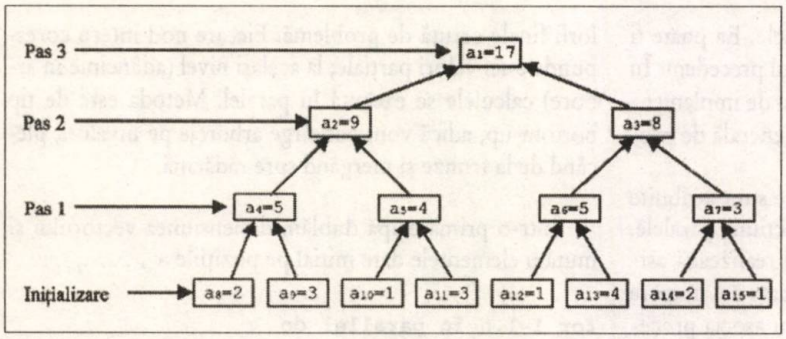
Clasa "principală" ArbBin conține câmpurile a (vectorul căruia i se aplică metoda), n (numărul elementelor vectorului) și m cu $n=2^m$. Metoda principală începe cu citirea elementelor vectorului, folosind metodele clasei input.

Pentru efectuarea calculelor {1}, adică pentru deplasarea elementelor vectorului pe pozițiile n..2n-1, este folosită clasa Tip1 ce extinde clasa Thread. Constructorul acesteia furnizează valorile n și i, iar metoda run prevede executarea calculului {1}. Sunt create n obiecte de tipul Tip1, pe baza cărora sunt lansate n fire de executare.

Să observăm că pentru a folosi modelul P-RAM (cu memorie comună) vectorul a a fost declarat cu **static** (variabilă de clasă), astfel încât să poată fi referit din afară prefixat cu numele clasei ArbBin în care a fost declarat și creat. De asemenea remarcăm utilizarea metodei join pentru așteptarea terminării firelor de executare lansate.

Pentru efectuarea prelucrării {2} se procedează în mod similar, folosind de această dată clasa Tip2, ce extinde și ea clasa Thread.

```
import java.util.*;
import java.io.*;
```




```
import input;

class Tip1 extends Thread {
  protected int i; protected int n;
  public Tip1(int i, int n) {
    this.i=i; this.n=n;
  }
  public void run() {
    ArbBin.a[n+i]=ArbBin.a[i];
  }
}
class Tip2 extends Thread {
  protected int i;
  public Tip2 (int i) { this.i=i; }
  public void run(){
    ArbBin.a[i]=ArbBin.a[2*i]+ArbBin.a[2*i+1];
  }
}
class ArbBin {
  static int a[]=new int[32]; static int n;
  static StreamTokenizer f =
    new StreamTokenizer(System.in);
  public static void main(String[] w) {
    int m=0; int i;
    System.out.println("m=");
    m = (int) input.read(f);
    System.out.println("Numerele: ");
    int n=1; for (i=0; i<m; i++) n *= 2;
    for (i=0; i<n; i++)
      a[i] = (int) input.read(f);
//-----------------------------------
    Tip1[] Ob1 = new Tip1[n];
   for (i=0; i<n; i++) Ob1[i] = new Tip1(i,n);
    for (i=0; i<n; i++) Ob1[i].start();
    try {for (i=0; i<n; i++) Ob1[i].join();}
    catch (InterruptedException e) {; }
//-----------------------------------
    Tip2[] Ob2 = new Tip2[n];
    int p = n/2 , u = n;
    for (int k=m-1; k>=0; k--) {
      for (i=p; i<u; i++)
        Ob2[i] = new Tip2(i);
for (i=p; i<u; i++)
  Ob2[i].start();
try {for (i=p; i<u; i++)
  Ob2[i].join();}
catch(InterruptedException e)
  {;}
for (i=p; i<u; i++)
  System.out.print(a[i]+" ");
  System.out.println();
  p=p/2; u=u/2;
}
System.out.println("Suma="+a[1]);
  }
}
```

Despre programarea concurentă

Am văzut că *programarea paralelă* pune accentul pe acțiunile ce trebuie executate de către fiecare procesor. Scopul programării paralele este de a elabora algoritmi care să speculeze existența mai multor procesoare pentru a micșora timpul de executare. Problemele sunt cele obișnuite din programarea secvențială, dar metodele de elaborare a programelor paralele vor fi evident diferite de cele din programarea secvențială.

Executarea concurentă a mai multor procese (fire de executare) aproape întotdeauna cere o anumită cooperare între ele, deci sunt necesare anumite mecanisme care să asigure comunicarea și sincronizarea între procese. Specificarea cooperării între procese în termenii mecanismelor disponibile, deci scrierea unor programe concurente, constituie obiectul *programării concurente*. Este nevoie de un alt mod de gândire, de alte "unelte", de altă înțelegere a noțiunii de program corect.

Esența programării concurente nu depinde de numărul procesoarelor fizice disponibile. Putem considera în permanență că dispunem de suficiente procesoare fizice pentru a atașa câte unul fiecăruia dintre procesele concurente. De asemenea putem considera că avem la dispoziție un singur procesor, care "alocă" proceselor intervale de timp aleatoare. Mai general, putem considera că este folosit *modelul aleator*, care constă din repetarea ciclică a următoarelor acțiuni de către fiecare procesor fizic:

1) alege aleator unul dintre procesele cărora nu îi este asociat un procesor fizic;
2) alege un interval de timp aleator și execută instrucțiuni ale sale.

Nu se va face nici o presupunere asupra vitezei de lucru a procesoarelor. Fiecare procesor trebuie gândit ca având un ceas logic propriu. Cu alte cuvinte, orice supoziții asupra raportului vitezelor a două procesoare, ca și asupra ritmicității executării instrucțiunilor în cod mașină din cadrul aceluiași proces, sunt neavenite (nu ne ocupăm aici de sistemele concurente în timp real).

În conformitate cu cele de mai sus, trebuie subliniat că programarea concurentă rezervă o serie de surprize informaticianului obișnuit cu programarea secvențială. Prezentăm în continuare câteva dintre ele.

- În programarea secvențială, următoarea secvență de instrucțiuni:

```
i=1; i=2;
```

este echivalentă cu a doua instrucțiune; prezența ei într-un program trezește îndoieli asupra profesionalismului autorului programului. În programarea concurentă secvența de mai sus nu are nimic redundant. Într-adevăr, în intervalul de timp dintre executarea celor două instrucțiuni (în cadrul aceluiași proces) este posibil ca celelalte procese să execute diferite instrucțiuni care să folosească efectiv faptul că un anumit interval de timp (asupra lungimii acestuia nu putem face nici o ipoteză) valoarea lui `i` este egală cu `1`.

- Efectul instrucțiunii:

```
if (i==1) j=j+i
```

în cazul în care valoarea lui `i` era `1` la începutul executării acestei instrucțiuni condiționale, nu constă neapărat în mărirea cu o unitate a valorii lui `j`, deoarece între momentul efectuării comparației și momentul efectuării atribuirii, valoarea lui `i` poate fi eventual modificată de un alt proces. Analog, dacă într-un proces apare secvența de instrucțiuni:

```
i=1; if (i==1) instr;
```

atunci, pe de o parte nu este neapărat adevărat că este îndeplinită condiția `i=1` din instrucțiunea **`if`**, iar pe de altă parte nu este neapărat adevărat că în momentul executării instrucțiunii *instr* valoarea lui `i` este egală cu `1`; în schimb afirmația "valoarea lui `i` a fost egală cu `1` la un moment de timp anterior" este adevărată și poate fi utilă (de exemplu în a demonstra că procesul are o anumită evoluție).

Este ușor de observat că aceste comportări "anormale" din punctul de vedere al programării secvențiale își au cauza în utilizarea de către mai multe procese a unei arii (comune) de memorie.

Programarea concurentă se mai deosebește de programarea secvențială și prin faptul că programele pot avea *comportare nedeterministă*: "la executări diferite pentru același set de date de intrare, rezultatele nu vor fi neapărat aceleași".

Astfel, dacă unul dintre procese execută instrucțiunile:

```
i=1; i=i+3;
```

iar un altul execută instrucțiunile:

```
i=5; i=i-2;
```

atunci printre valorile ce pot rezulta în final pentru `i` se numără de exemplu `3`, `4`, `6`.

Programarea concurentă este menită să rezolve interacțiunile dintre procese. Atenția principală este îndreptată asupra modalităților în care pot fi realizate interacțiunile dorite și mai puțin asupra a ceea ce realizează programele secvențiale ce constituie fiecare proces în parte.

Natura problemelor este diferită de cea întâlnită în programarea secvențială. Sunt tratate în special probleme de simulare a unor situații concurente reale și nu probleme care urmăresc obținerea unui rezultat numeric.

Iată două exemple tipice de probleme întâlnite în programarea concurentă:

1) ***Problema rezervării biletelor:*** *Fiecare terminal al unei rețele de calculatoare este plasat într-un punct de vânzare a biletelor pentru un anumit spectacol. Se cere să se găsească o modalitate de a evita vânzarea a două bilete pentru același loc.*
2) ***Problema Producător - Consumator:*** *Producătorul fabrică articole de un anumit tip și le plasează pe o bandă. Consumatorul ia câte un obiect de pe bandă și îl "consumă". Se cere să se simuleze aceste activități, evitându-se încercările producătorului de a plasa un articol pe bandă când aceasta este plină, precum și încercările consumatorului de a lua un obiect de pe bandă când aceasta este goală (dificultatea constă deci în realizarea unei sincronizări).*

Vom numi *secțiune critică* o secvență de instrucțiuni a căror executare trebuie să se supună următoarei reguli: *în momentul în care un proces* `P` *începe executarea primei instrucțiuni din secvență, nici un alt proces nu poate executa instrucțiuni din secvență înainte de terminarea executării ultimei instrucțiuni din secvență de către procesul* `P`. În aceste condiții spunem că are loc o *excludere reciprocă* (cel mult un proces se poate afla într-o secțiune critică). Importanța și rolul secțiunilor critice vor fi puse în evidență pe un nou exemplu.

Problema grădinilor ornamentale (atribuirii multiple). *Intrarea în grădinile ornamentale ale unui oraș oriental se face prin mai multe porți. Se pune problema ținerii evidenței numărului total de persoane care au intrat în grădini.*

O încercare de rezolvare este următoarea:
- este introdusă variabila globală `total`, a cărei valoare curentă reprezintă numărul persoanelor ce au intrat în grădini;
- fiecărei porți i se asociază în mod natural un proces, ce prevede ca în mod repetat valoarea variabilei `total` să fie mărită cu o unitate, corespunzător intrării unei persoane pe poarta respectivă.

Încercarea de mai sus nu este corectă: în final `total` poate avea o valoare diferită de cea așteptată; mai mult, la executări diferite ale programului ce transpune această încercare de rezolvare, se pot obține rezultate diferite. Explicația constă în primul rând în faptul că instrucțiunea `total=total+1` este divizibilă (nu este o acțiune "atomică"), iar în al doilea rând în modul de executare al programelor concurente. Într-adevăr, instrucțiunea

```
total=total+1
```

este în realitate o secvență formată din trei instrucțiuni în cod mașină:

```
LOAD total,r  { încarcă total în registrul r al CPU    }
INC r         { incrementează valoarea lui r cu 1      }
STORE r,total { salvează valoarea registrului în total }
```

Pentru exemplificare, să considerăm cazul a două porți. Fie `r1` și `r2` registrele unității centrale folosite de cele două procese corespunzătoare. În cazul în care pe fiecare poartă intră trei persoane, este posibilă (printre multe altele) următoarea succesiune de acțiuni:

	total	r1	r2
	0		
p1 încarcă		0	
p1 incrementează		1	
p2 încarcă			0
p2 incrementează			1
p2 salvează	1		
p2 încarcă			1
p2 incrementează			2
p2 salvează	2		
p1 salvează	1		

urmată de o succesiune analoagă în care se interschimbă `p1` cu `p2`. Rezultatul final va fi `2`, mai mic chiar decât numărul persoanelor ce intră prin fiecare poartă!

Să observăm că dacă instrucțiunea `total=total+1` (cele trei instrucțiuni mașină în care se descompune) ar forma o secțiune critică, rezultatul ar fi totdeauna corect.

Rezolvarea excluderii reciproce

Vom rezolva în acest paragraf *problema grădinilor ornamentale*. Reamintim enunțul ei: *Se cere să se țină evidența numărului de persoane ce intră pe cele* `n` *porți de acces în grădinile ornamentale.*

Programul de mai jos transpune ideea de rezolvare de mai sus, adică lansăm `n` fire de executare care prevăd fiecare mărirea de un număr dat de ori a unei variabile comune `total`. Pentru a simula modul în care se realizează efectiv incrementarea lui `total` cu o unitate (este utilizat un registru), este folosită variabila suplimentară `temp`. În plus, pentru a accentua caracterul nedeterminist, este folosită metoda `yield` cu zero argumente a clasei `Thread`. Efectul apelării lui `yield` din cadrul unui fir de executare este următorul: se dă o șansă unui alt fir de executare (care poate fi și cel curent) de a deveni activ.

```
import java.util.*;
class Tip extends Thread {
  static int total=0; int i;
  Tip(int i) { this.i=i; }
  public void run() {
    int temp;
    for (int j=1; j<=10; j++) {
      System.out.print(" " + i);
      yield(); temp=total;
      yield(); temp++;
      yield(); total=temp;
    }
  }
}
```

```
class Gradini {
  public static void main(String[] w) {
    Tip[] Ob = new Tip[5];
    for(int i=0;i<5;i++) Ob[i]=new Tip(i);
    for(int i=0;i<5;i++) Ob[i].start();
    try{for(int i=4;i>=0;i--) Ob[i].join();}
    catch (InterruptedException e) { }
    System.out.println("Total="+Tip.total);
  }
}
```

În urma executării programului de mai sus, doar în mod excepțional apare la ieșire valoarea `50` preconizată. Mai mult, valoarea tipărită la ieșire poate să difere de la o executare la alta. Explicația constă în faptul că `total` este o variabilă accesată simultan de mai multe fire de executare și în neatomicitatea operațiilor prevăzute de metoda `run` a clasei `Tip`.

Pentru a realiza *excluderea reciprocă* asupra operației de incrementare a variabilei comune `total`, vom folosi facilitatea din limbajul Java de a declara o metodă cu modificatorul **`synchronized`**.

Regula este următoarea: dacă mai multe fire de executare apelează o metodă declarată cu modificatorul **`synchronized`** prin intermediul *aceluiași* obiect, având ca tip clasa ce conține metoda respectivă, atunci la fiecare moment de timp cel mult un fir de executare este în curs de a executa acea metodă. Dacă un fir de executare este în curs de a executa metoda, el o va executa până la capăt; orice alt fir de executare ce invocă metoda, va fi blocat. Când un fir de executare termină metoda, atunci unul dintre firele de executare blocate va fi deblocat și va deveni el cel care deține "controlul" (exclusiv) asupra metodei.

În programul de mai jos firele de executare apelează metoda `numara`, declarată cu **`synchronized`**, a clasei `Contor` prin intermediul aceluiași obiect `C` de tipul clasei `Contor`.

```
import java.util.*;

class Contor {
  static int total=0;
  synchronized void numara() {
    int temp;
    Thread.yield(); temp=total;
    Thread.yield(); temp++;
    Thread.yield(); total=temp;
  }
}

class Tip extends Thread {
  int i; Contor C;
  Tip(int i, Contor C)
  {
    this.i=i;
    this.C=C;
  }
```

```
  public void run() {
    int temp;
    for (int j=0; j<10; j++) {
      C.numara(); System.out.print(i+"\t");
    }
  }
}

class Gradinil {
  public static void main(String[] w) {
    Contor C = new Contor();
    Tip[] Ob = new Tip[5];
    for (int i=0; i<5; i++)
      Ob[i] = new Tip(i,C);
    for (int i=0; i<5; i++) Ob[i].start();
    try {for (int i=4; i>=0; i--)
      Ob[i].join();}
    catch (InterruptedException e) { }
    System.out.println("Total = "+C.total);
  }
}
```

Interfața Runnable

Interfața `Runnable` apare în pachetul `java.lang` (deci nu necesită o specificare explicită prin **`import`**).

Această interfață permite executarea unui cod ca parte a unui fir de executare activ. Ea cuprinde o singură metodă și anume:

```
public void run()
```

care este executată când firul este activ. Metoda `run` nu prevede nici o acțiune și trebuie extinsă în clasa ce implementează `Runnable`.

Menționăm că printre diferitele clase care implementează `Runnable` se află și clasa `Thread`.

Utilitatea interfeței `Runnable` constă în primul rând în însuși faptul că este o interfață: o clasă oarecare poate implementa `Runnable` și extinde o altă clasă (pe când o clasă ce extinde `Thread` nu mai poate extinde vreo altă clasă).

Modul tipic de utilizare este următorul:

```
class Execut implements Runnable {
  ...
  public void run {
    ...
  }
  ...
}

class Clasa {
  ...
  Execut Ob=new Execut(...);
  Thread Fir=new Runnable(Ob);        // *
  Fir.start();                        // **
  ...
}
```

Astfel, este creat (*) și apoi este lansat (**) firul de executare `Fir`.

Pentru ilustrare, reluăm exemplul 1 din acest capitol, utilizând interfața `Runnable` în loc de clasa `Thread`.

```
class Execut implements Runnable {
  int k=0; boolean continua=true;
  public void run() {
    while (continua)
      System.out.println(" "+k++);
    System.out.println("Fir");
  }
  public void terminare() { continua=false; }
}

class RunFir {
  public static void main(String[] arg)
      throws java.io.IOException {
    Execut ob = new Execut();
    Thread fir = new Thread((Runnable) ob);
    fir.start();
    while((char) System.in.read() != '\n');
    ob.terminare();
    System.out.println("main");
  }
}
```

O soluție pentru problema Producător-Consumator

Trecem în continuare la un alt mod de sincronizare, ce are ca model *problema Producător - Consumator*. Reamintim enunțul ei: *Un producător și un consumator își desfășoară în același timp activitatea, folosind în comun o bandă de lungime* `lung`. *Producătorul produce obiecte și le plasează la un capăt al benzii. Consumatorul ia câte un obiect de la celălalt capăt al benzii și îl consumă.*

Dificultatea problemei constă în faptul că producătorul și consumatorul pun/iau obiecte de pe bandă în ritmuri imprevizibile, ceea ce poate conduce la următoarele situații limită:

- producătorul încearcă să pună un obiect în banda plină;
- consumatorul încearcă să ia un obiect din banda vidă.

Clasa `Time` furnizează metoda `delay` care întoarce un număr real cuprins între 0 și 100. Ea va fi folosită pentru a accentua nedeterminismul, operațiile de punere și luare pe/de pe bandă fiind întârziate un timp aleator.

Clasa `Banda` ține evidența cozii corespunzătoare benzii: vectorul `coada` are lungimea `lung`, primul său element este pe poziția `p`, ultimul pe poziția `u`, iar numărul elementelor de pe bandă este `n`. Metoda `ia` realizează extragerea unui element de pe bandă, iar metoda `pune` corespunde introducerii unui nou element pe bandă; elementele sunt caractere. Metodele `ia` și `pune` sunt declarate cu modificatorul **`synchronized`**.

Obiectul `P` de tipul `Prod`, clasă ce implementează interfața `Runnable`, lansează un fir de executare corespun-

zător producătorului; sunt produse literele mici ale alfabetului și se încearcă plasarea lor pe bandă prin invocarea metodei pune.

Obiectul c de tipul Cons, clasă ce implementează și ea interfața Runnable, lansează un fir de executare corespunzător consumatorului; prin invocarea metodei ia se încearcă preluarea de caractere de pe bandă, până când s-a luat 'z'.

Cele două fire de executare folosesc același obiect b de tipul Banda pentru invocarea metodelor ia și pune.

Sincronizarea este asigurată prin folosirea metodelor:

- wait(), care face ca firul de executare curent să fie blocat; în același timp este ridicată excluderea reciprocă;
- notify(), care cedează controlul unui alt fir de executare ce este blocat; în același timp este ridicată excluderea reciprocă. Firul curent este blocat. Dacă nici un alt fir de executare nu așteaptă (nu este blocat), efectul este nul.

```
import java.util.*;

class Time extends Random {
  float delay() {
    return 100.0f*nextFloat();
  }
}

class Banda {
  static int lung=3, n=0, p=0, u=lung-1;
  static char[] coada = new char[lung];
  static String out = "";
  static Time ObT = new Time();
  synchronized char ia() {
    char c;
    while (n == 0) {
      try { wait(); }
      catch (InterruptedException e) {}
    }
    c = coada[p]; p = (p+1)%lung; n--;
    out += "  I" + c; notify(); return c;
  }

  synchronized void pune(char ch) {
    while (n == lung) {
      try { wait(); }
      catch (InterruptedException e) {}
    }
    u=(u+1)%lung; coada[u]=ch; n++;
    out+=" P"+ch; notify();
  }
}

class Prod implements Runnable {
  static Banda b;
  Prod(Banda bb) { b = bb; }
  public void run() {
    float d; char ch;
    for (ch='a'; ch<='z'; ch++) {
      b.pune(ch);
      try {d=Banda.ObT.delay();
           sleep((int)d);
          }
      catch (InterruptedException e) {} }
  }
}

class Cons implements Runnable {
  static Banda b;
  Cons(Banda bb) {b=bb;}
  public void run(){
    float d; char ch;
    do {
      ch=b.ia();
      try{ d=Banda.ObT.delay();
           sleep((int)d);
          }
      catch (InterruptedException e) {} }
    while (ch != 'z');
  }
}

class PC {
  public static void main(String[] w) {
    Banda b=new Banda();
    Prod P=new Prod(b);
    Cons C=new Cons(b);
    Thread FirP=new Thread(P);
    FirP.start();
    Thread FirC=new Thread(C);
    FirC.start();
    try { FirP.join(); FirC.join(); }
    catch (InterruptedException e) {}
    System.out.println(Banda.out);
  }
}
```

Este de asemenea utilizat un șir de caractere out în care sunt adăugate semnalizările corespunzătoare punerii unui element pe bandă, respectiv preluării unui element de pe bandă.

Clișeul standard de folosire a metodei wait este următorul: **while** (! *condiție*) wait(); și a fost folosit ca atare în programul de mai sus deși, prin natura problemei, puteam înlocui **while** cu **if**.

După cum se vede, metodele wait și notify pot lansa excepția cu numele: InterruptedException.

Mai menționăm că metoda wait admite și varianta cu un argument, ce este un număr de tip long, reprezentând durata maximă de așteptare.

Prof. dr. Horia Georgescu este cadru didactic la Universitatea București; poate fi contactat prin e-mail la adresa: hg@colegiu.cs.unibuc.ro

Pagini WEB cu HTML 4.0

Mihai Scorțaru

Programarea în Internet reprezintă unul dintre cele mai interesante domenii din cadrul IT&C. Dacă doriți să aflați cum puteți îmbunătăți calitatea paginilor dumneavoastră, citiți acest al cincilea episod al serialului despre HTML. Veți afla cum puteți include script-uri JavaScript și VBScript, applet-uri Java, precum și controale ActiveX. Vor fi prezentate și noi facilități oferite de specificațiile HTML 4.0.

Limbajul HTML se bazează pe un limbaj de descriere a redactării paginilor și documentelor numit *SGML (Standard Generalized Markup Language)*. Acest limbaj oferă multe proprietăți remarcabile, dar impune limitări aplicate anumitor domenii. Pentru a trece peste acestea, proiectanții au apelat la noi limbaje de programare de tip ***script*** pentru a putea dezvolta în continuare proprietățile paginilor Web. Aceste noi extensii reprezintă modalitatea prin care pot fi proiectate pagini Web care să corespundă tehnologiei de ultimă oră.

Am folosit în paragraful anterior noțiunea de limbaj de tip *script*. Pentru cei nefamiliarizați cu această noțiune trebuie precizat faptul că limbajele de tip *script* sunt niște limbaje de programare simple, de o complexitate scăzută, motiv pentru care sunt des folosite pentru programarea în Web.

JavaScript

Vom începe cu cel mai utilizat limbaj de tip script la ora actuală, și anume JavaScript. Majoritatea celor care utilizează rețeaua Internet, confundă limbajul de programare JAVA cu mult mai puțin complexul limbaj JavaScript. Acesta din urmă este un limbaj foarte simplu care permite definirea comportamentului unor elemente dintr-o pagină Web.

Limbajul JavaScript seamănă cu fișierele BAT de comenzi secvențiale din DOS sau cu scripturile UNIX de tip *shell*. Nu ne propunem să prezentăm în detaliu acest limbaj (s-ar putea scrie o carte de câteva sute de pagini despre acest subiect), ci vom face doar o prezentare generală a limbajului.

Vom începe cu tradiționala aplicație care afișează mesajul "Hello World!", programată de data aceasta în JavaScript. Fișierul HTML corespunzător este următorul:

Pagini Web despre JavaScript

JavaScriptMadeEasy:
http://www.calpoly.edu/~dpopeney/javascript.html
JavaScript:
http://javascript.developer.com/
JavaScriptPlanet:
http://www.geocities.com/SiliconValley/7116/
NetscapeJavaScriptGuide:
http://developer.netscape.com/library/documentation/ communicator/jsguide/index.html
JavaScriptLearningResources:
http://www.mcp.com/sams/books/235-8/hotlist.htm

```
<html>
<head>
  <title> Hello World! </title>
  <script language="JavaScript">
    <!-- sample JavaScript
    alert ('Hello World!')
    // sfarsitul script-ului-->
  </script>
</head>
<body>
  Aceasta este pagina Web care contine prima
  aplicatie JavaScript.
</body>
</html>
```

După cum observați, tag-ul HTML pentru introducerea unui script este **<SCRIPT>**. Acesta are un parametru LANGUAGE a cărui valoare indică limbajul de tip script folosit. Tag-ul **<SCRIPT>** apare în interiorul tag-ului **<HEAD>** și nu în interiorul tag-ului **<BODY>**, așa cum era cazul majorității tag-urilor HTML care realizau diferite acțiuni.

Browserele vechi vor ignora tag-ul **<SCRIPT>**, dar nu vor ignora ceea ce va apărea între opening tag-ul și closing tag-ul acestei comenzi. De aceea, o practică des întâlnită este înglobarea întregului script într-un comentariu pentru a fi ignorat de browserele care nu suportă script-uri. Exemplul anterior ilustrează modul în care sunt introduse comentarii în limbajul HTML și modul în care aceste comentarii sunt introduse în cadrul limbajului JavaScript. Efectul acestui script este vizualizat în imaginea următoare.

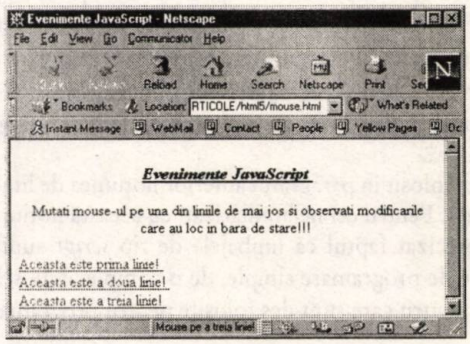
Observați că zona client a ferestrei browserului nu este actualizată la apariția ferestrei generate de aplicația JavaScript, această actualizare având loc numai după apăsarea butonului **OK**. După apăsarea acestui buton fereastra browserului va arăta astfel:

Apăsarea butonului duce la dispariția ferestrei generate de aplicația JavaScript.

Limbajul JavaScript permite și specificarea unor răspunsuri la diferite acțiuni, cum ar fi deschiderea unei pagini, deplasarea mouse-ului într-o anumită zonă a ecranului etc. În următorul exemplu se modifică ceea ce apare în bara de stare a browserului în funcție de zona în care se află cursorul mouse-ului.

```
<html>
<head>
  <title> Evenimente JavaScript </title>
</head>
<body>
  <font size = 4>
  <p align="center">
  <br><b><i><u>
    Evenimente JavaScript
  </b></i></u>
  </p>
  </font>
  <center>
    Mutati mouse-ul pe una din liniile de mai
    jos si observati modificarile care au loc
    in bara de stare!!!
  </center>
  <p>
  <a href = "linia #1.html"
  onMouseOver="self.status='Mouse pe prima
                    linie!'; return true">
  Aceasta este prima linie! </a>
  <br>
  <a href = "linia #2.html"
  onMouseOver="self.status='Mouse pe a doua
                    linie!'; return true">
  Aceasta este a doua linie! </a>
  <br>
  <a href = "linia #3.html"
  onMouseOver="self.status='Mouse pe a treia
                    linie!'; return true">
  Aceasta este a treia linie! </a>
</body>
</html>
```

În cazul în care veți muta mouse-ul deasupra celui de-al treilea link, pagina afișată va arăta astfel:



Dacă veți muta cursorul mouse-ului deasupra primului link, bara de stare va fi modificată, ea afișând mesajul: `Mouse pe prima linie`. Situația este asemănătoare cazului în care veți muta cursorul mouse-ului deasupra celei de-a doua linii.

În exemplul anterior a fost folosit tag-ul **<CENTER>** pentru centrarea anumitor elemente care apar în pagină. În primul episod al serialului am spus că acest tag a fost oficial eliminat din specificațiile HTML. Totuși, el a fost reintrodus în noile specificații HTML 4.0.

Cu ajutorul limbajului JavaScript pot fi gestionate mai multe evenimente diferite. Cele mai importante astfel de evenimente sunt: *blur*, *click*, *change*, *focus*, *load*, *mouseover*, *select*, *submit* și *unload*. Pentru a "sesiza" aceste evenimente, browserul trebuie să ofere niște funcții speciale (denumite *handler-e*) care să indice momentul în care are loc respectivul eveniment. Handler-ele corespunzătoare evenimentelor prezentate anterior sunt `onMouseOver`,

onBlur, onSelect, onLoad, onFocus, onSubmit, onClick, onChange și onUnload. În tabelul alăturat vă prezentăm situațiile în care apar aceste evenimente.

După cum observați multe dintre aceste evenimente semnalează acțiuni ce au loc în cadrul paginilor care conțin formulare. Mai multe informații despre formularele HTML veți găsi în episodul următor al serialului nostru.

Continuăm cu prezentarea unor aplicații JavaScript mai complexe. Să presupunem că dorim ca o imagine statică să fie înlocuită cu una animată în momentul în care cursorul mouse-ului este mutat deasupra ei și să revină la forma inițială în momentul în care cursorul mouse-ului este mutat în altă zonă a ecranului. Fișierul sursă este următorul:

Evenimente

Evenimentul	Apare atunci când vizitatorul:
blur	- elimină controlul de intrare de pe un element al unui formular
click	- efectuează un click pe un element al unui formular sau pe un link
change	- modifică valoarea unui text, zonă de text sau element de selectare
focus	- atribuie unui element de formular control de intrare
load	- comandă browserului încărcarea paginii respective
mouseover	- deplasează cursorul mouse-ului deasupra unui link
select	- selectează câmpul de intrare al unui element de formular
submit	- transmite un formular
unload	- părăsește pagina respectivă

```
<HTML>
  <TITLE> Imagini </TITLE>
  <BODY BGCOLOR=white>
  <SCRIPT LANGUAGE="JavaScript">
  <!--
  imganimata = new Image();
  imgstatica = new Image();
  imganimata.src = 'animat.gif';
  imgstatica.src = 'static.gif';
  function animatie(imagine) {
    imagesrc=eval(imagine+'animata.src');
    document[imagine].src=imagesrc;
  }
  function imagine_statica(imagine) {
    imagesrc=eval(imagine+'statica.src');
    document[imagine].src=imagesrc;
  }
  // sfarsitul script-ului          -->
  </SCRIPT>
  <CENTER>
  <a href=clicked.html
  OnMouseOver="animatie('img');return true;"
  OnMouseOut="imagine_statica('img');
                                  return true;">
  <img src=static.gif alt="Imagine"
                    border=0 name="img"</a>
  </CENTER>
</BODY>
</HTML>
```

Datorită faptului că în paginile revistei nu putem reda animația care apare în fereastra browserului, nu vom mai prezenta o captură a acestei ferestre. Putem să vă spunem că la încărcarea paginii va apărea o imagine statică, imagine ce se va "anima" în momentul în care cursorul mouse-ului este deplasat deasupra ei.

Să analizăm acum fișierul HTML anterior. Pentru a realiza o modificare dinamică a unei imagini (adică modificare în urma unui eveniment) trebuie mai intâi să denumim respectiva imagine. Pentru aceasta am folosit un atribut al tag-ului **<IMG>** numit **NAME** prin care am atribuit respectivei imagini numele img. Instrucțiunea JavaScript:

```
document[imagine].src = imagesrc;
```

utilizează numele respectivei imagini (img) pentru a stabili care dintre elementele paginii trebuie modificate. Elementul esențial, în acest exemplu, este modalitatea prin care se face legătura dintre evenimentele de animare, respectiv de revenire la imaginea statică, și evenimentele Web onMouseOver și onMouseOut. În cazul apariției evenimentului onMouseOver este apelată funcția animatie, iar în cazul apariției evenimentului onMouseOut este apelată funcția imagine_statica. Ambele funcții trebuie să determine numele fișierului în care este păstrată imaginea care trebuie afișată. Acest lucru este realizat cu ajutorul instrucțiunii:

```
imagesrc = eval(imagine + 'animata.src');
```

pentru funcția animatie, respectiv cu ajutorul instrucțiunii:

```
imagesrc = eval(imagine + 'statica.src');
```

pentru funcția imagine_statica.

În cadrul acestor instrucțiuni, la valoarea parametrului imagine (funcția este apelată cu valoarea img pentru acest parametru) este concatenat șirul 'animata.src', respectiv 'statica.src', obținându-se pentru variabila imagesrc valoarea corespunzătoare câmpului imgstatica.src, respectiv imganimata.src. Imganimata și imgstatica sunt două obiecte instanță ale clasei Image care sunt declarate în primele linii ale script-ului. Obiectele care fac parte din clasa Image conțin un câmp (src) ce indică fișierul în care se află imaginea corespunzătoare respectivului obiect.

Pentru cei familiarizați cu conceptul de *programare orientată pe obiecte* nu ar trebui să existe probleme în înțelegerea explicațiilor de mai sus. Pentru a explica, pentru ceilalți, conceptul de programare orientată pe obiecte nu ar fi suficiente paginile revistei. Cei care doresc explicații suplimentare sunt rugați să consulte documentații despre acest concept sau să ia legătura cu redacția GInfo.

O altă facilitate remarcabilă oferită de JavaScript este posibilitatea afișării unui mesaj derulant în bara de stare. Am văzut deja cum poate fi stabilită valoarea stării în funcție de un anumit eveniment, prin modificarea valorii *window.status*, dar următorul script vă demonstrează cum puteți obține efecte mai spectaculoase.

```
<HTML>
  <TITLE>
    Mesaj derulant
  </TITLE>
  <SCRIPT LANGUAGE="JavaScript"
  <!--

  function StartScroll() {
    msg="          Hello World!          "
    ScrollMessage();  // inceperea derularii
  }

  function ScrollMessage(){
    window.status = msg;
          // seteaza mesajul la sirul curent
    msg=msg.substring(1,msg.length)+
                              msg.substring(0,1);
        // produce un nou sir prin permutare
        // la stanga cu o pozitie
    timer=setTimeout("ScrollMessage()", 100);
          // intervalul de timp (in miimi de
          // secunda) care trebuie sa treaca
          // pana la o noua actualizare a
          // barei de stare.
  }
  StartScroll();          // incepe derularea.
  // sfarsitul scriptului  -->
</SCRIPT>

<BODY BGCOLOR=white>
  <CENTER>
  <font size = 5>
    S.C. Electric 21 Service S.R.L.
    Cluj-Napoca
  </font>
  <p>
  <font size = 6>
    Mihai Scortaru
  </font>
  <br>
  <b> <i> Technical Manager </b> </i>
  <p>
  <table border = 0>
  <tr>
  <td valign='top'>
    <font color = "red" size = 4>
      Phone:
    </font>
  </td>
  <td>
    <font color = "red" size = 4>
      +40-92-327635 <br>
      +40-64-419835
    </font>
  </td>
  </tr>
  </table>
  </font>
  </CENTER>
</BODY>
</HTML>
```

Imaginea următoare prezintă un instantaneu al modului în care se execută acest script, dar, pentru a vedea cum funcționează el în realitate trebuie să îl vizionați pe ecranul unui calculator.

Script-ul începe să se execute atunci când programul de navigare întâlnește apelul funcției `StartScroll()`. În cadrul acestei funcții se inițializează șirul de caractere care reprezintă mesajul și se apelează funcția `ScrollMesage()`. Această a doua funcție modifică bara de stare, permută cu o poziție spre stânga elementele șirului, așteaptă o zecime de secundă și apoi se autoapelează.

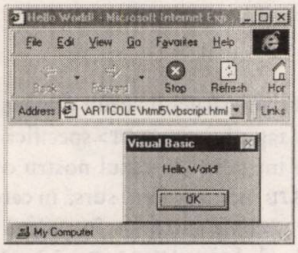
Ne oprim aici cu prezentarea limbajului JavaScript deoarece nu ne-am propus descriem limbajul, ci să punctăm doar câteva caracteristici importante. Ne vom mai întâlni cu acest limbaj în cadrul episodului care va prezenta formularele HTML, deoarece prelucrarea acestora poate necesita anumite elemente de programare.

Visual Basic Script

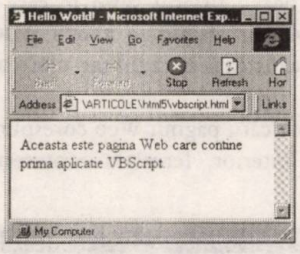
JavaScript este un limbaj de tip script foarte puternic dar diferă, în mod esențial, de orice limbaj de programare cunoscut de majoritatea programatorilor. Chiar și diferențele față de JAVA sunt semnificative. Pentru a suplini această lipsă, firma Microsoft oferă limbajul Visual Basic Script pentru Internet Explorer - VBScript - ca o alternativă la JavaScript. VBScript este un limbaj simplu și rapid care suportă o parte a sintaxei limbajului *VBA* (*Visual Basic for Applications*) și care a fost special proiectat pentru a fi folosit la scrierea aplicațiilor pentru Internet. Vom prezenta o aplicație VBScript simplă:

```
<html>
<head>
  <title> Hello World! </title>
<script language="VBScript">
  <!--
  MsgBox "          Hello World!          "
  -->
</script>
</head>
<body>
   Aceasta este pagina Web care contine
   prima aplicatie VBScript.
</body>
</html>
```

Efectul este asemănător celui observat în cazul primei aplicații JavaScript prezentate. La început va apărea o imagine asemănătoare celei prezentate în continuare:



Pentru ca încărcarea paginii să continue, trebuie apăsat butonul OK. În urma apăsării acestui buton, fereastra browserului va arăta astfel:



În mod normal nu ar trebui să fie necesare explicații suplimentare cu privire la modul în care este executat acest script, asemănarea cu script-ul JavaScript prezentat inițial fiind foarte mare, motiv pentru care vom face doar câteva precizări referitoare la VBScript.

Cu toate că limbajul VBScript este mult mai ușor de utilizat decât limbajul JavaScript, el nu poate fi (încă?) folosit pe scară largă de proiectanții de pagini Web. Motivul îl reprezintă incompatibilitatea dintre acest limbaj și browserul Netscape Navigator care este încă, cel mai folosit și, probabil, cel mai bun browser la ora actuală. Din această cauză script-urile scrise, folosind acest limbaj nu vor avea efect decât dacă browserul folosit este Internet Explorer, ceilalți utilizatori neputând vedea efectele script-urilor respective decât dacă Netscape va furniza o versiune a browserului său care să recunoască și aceste script-uri. Dar cum această firmă nu are nici un interes să faciliteze impunerea limbajului oferit de firma rivală Microsoft, s-ar putea ca suportul Netscape pentru VBScript să întârzie destul de mult.

Probabil cititorii acestor rânduri știu că provider-ul de servicii Internet *AOL* (*America OnLine*) a cumpărat la sfârșitul lunii noiembrie a anului trecut firma Netscape, obținând astfel toate drepturile asupra pachetului de programe Netscape Communicator. La puțin timp după acest eveniment firma Sun Microsystems a obținut de la AOL drepturile de comercializare a pachetelor de software produse de Netscape și dedicate întreprinderilor. Se prefigurează apariția unei alianțe a marilor firme din domeniul IT&C împotriva firmei Microsoft, alianță din care fac parte, până în prezent firmele Sun, AOL și Netscape. Această alianță este denumită de presa de specialitate gruparea *ABM* (*Anyone But Microsoft - Oricine în afară de Microsoft*), fapt ce indică clar intenția firmelor din domeniul IT&C de a stopa avansul Microsoft în acest domeniu. Ce legătură au toate acestea cu limbajul VBScript? Foarte simplu! Gruparea ABM va încerca prin orice mijloace să obstrucționeze impunerea oricărui produs Microsoft, și deci și a limbajului VBScript, motiv pentru care probabil va trece destul de multă vreme până la apariția suportului Netscape pentru VBScript. O grupare care ar fi putut opri acest conflict este Consorțiul World Wide Web (W3), grupare care se ocupă cu definirea și standardizarea specificațiilor HTML. Din nefericire, acest consorțiu a hotărât să nu se implice în lupta dintre Microsoft și "restul lumii", indicând numai modul în care un limbaj de tip script trebuie introdus într-o pagină Web și nerecomandând un limbaj anume. Deși ar fi fost mult mai bine pentru proiectanții de pagini să afle că trebuie să se mulțumească cu facilitățile JavaScript, W3 a hotărât că viitoarele browsere vor putea oferi suport pentru diferite limbaje de tip script în momentul în care acestea "ajung la maturitate". Această poziție nu face decât ca *Bătălia pentru Internet* dintre Microsoft și ABM să continue.

Se pot afla mai multe despre Visual Basic Script vizitând una dintre următoarele pagini Web:

http://www.vbscripts.com
http://www.microsoft.com/VBASIC/.

Java

Limbajul de programare Java, elaborat de firma Sun Microsystems, a reprezentat o adevărată revoluție în programarea pentru Internet. Java este un limbaj de programare complex, orientat pe obiecte și bazat pe puternicul limbaj C++. Cea mai importantă caracteristică a aplicațiilor Java este *portabilitatea*, o aplicație Java putând rula, teoretic, pe orice platformă pentru care a fost scris un interpretor pentru mașina virtuală Java. Nu ne propunem să facem o prezentare a limbajului Java (chiar pe paginile GInfo există o prezentare a acestui limbaj) totuși nu pu-

tem trece mai departe fără a arăta cum se afișează în Java mesajul "Hello World!"

```
class HelloWorld {
  public static void main (String args[]) {
    System.out.println("Hello World!");
  }
}
```

Nu credem că ar putea exista neclarități cu privire la exemplul anterior. Pentru cei care doresc să afle mai multe despre programarea în Java le recomandăm să urmeze una din următoarele căi:

- studiul articolelor domnului Prof. Dr. Horia Georgescu apărute în GInfo;
- studiul unui ghid de programare Java aflat pe Internet; un excelent astfel de ghid se află la adresa http://sunsite.unc.edu/javafaq/javatutorial.html.
- studiul unei cărți care prezintă acest limbaj; la editura Computer LIBRIS Agora din Cluj-Napoca a apărut cartea *Limbajul Java* scrisă de Irina Athanasiu, Bogdan Costinescu, Octavian Andrei Drăgoi și Florentina Irina Popovici.

Un program Java, asemănător celui prezentat anterior, nu poate fi inclus într-o pagină Web, deoarece în aceste pagini nu pot apărea decât un anumit tip de aplicații Java, numite *applet*-uri. Un *applet* este *o colecție de clase Java care se execută prin interpretarea codului obiect de către mașina virtuală dintr-un browser*. Una dintre aceste clase este clasa principală, iar fișierul care conține codul ei va fi invocat în cadrul documentului HTML. Această clasă principală trebuie să extindă întotdeauna o clasă specială numită `Applet`.

După cum v-ați obișnuit deja, vom începe cu prezentarea unui applet care afișează mesajul "Hello World!". Codul sursă pentru acest applet este următorul:

```
import java.applet.*;
import java.awt.*;
public class HelloWorld extends Applet{
  public void paint (Graphics graphic) {
    graphic.drawString("Hello World!",10,10);
  }
}
```

În urma compilării fișierului prezentat anterior **`HelloWorld.java`** va rezulta un fișier care va conține codul obiect al applet-ului **`HelloWorld.class`**. Acesta este fișierul care va fi referit în cadrul documentului HTML.

Până la versiunea HTML 3.2 un applet era introdus într-o pagină Web folosind tag-ul **`<APPLET>`**. Specificațiile HTML 4.0 înlocuiesc acest tag cu tag-ul **`<OBJECT>`**, acest nou tag având diferite facilități suplimentare. Recomandăm folosirea tag-ului **`<OBJECT>`**, deoarece tag-ul **`<APPLET>`** este pe cale de a fi eliminat din limbajul HTML.

Vă prezentăm în continuare fișierul HTML care include într-o pagină WEB de două ori applet-ul **`HelloWorld.class`**, prima dată utilizând **`<OBJECT>`**, iar apoi utilizând tag-ul **`<APPLET>`**.

```
<html>
<head>
  <title>
    Applet
  </title>
</head>
<body>
  <object codetype="application/octet-stream"
    classid="java:HelloWorld.class"
    width = 70 height = 30>
  </object>
  <applet code="HelloWorld.class"
    width = 70 height = 30>
  </applet>
</body>
</html>
```

Să analizăm acum structura celor două tag-uri. Atributul **`CODETYPE`** al tag-ului **`<OBJECT>`** specifică tipul obiectului care trebuie introdus (în cazul nostru o aplicație), iar atributul **`CLASSID`** indică fișierul sursă în care se află codul obiectului care trebuie introdus în pagină (în exemplul nostru este vorba de fișierul **`HelloWorld.class`** care conține codul compilat al applet-ului scris în Java). Atributele **`WIDTH`** și **`HEIGHT`** reprezintă lungimea, respectiv înălțimea, zonei de ecran care va fi rezervată applet-ului respectiv. Structura tag-ului **`<APPLET>`** este mult mai simplă, atributul **`CODE`** specificând fișierul sursă în care se află codul applet-ului Java, iar atributele **`WIDTH`** și **`HEIGHT`** având aceeași semnificație ca și în cazul tag-ului **`<OBJECT>`**.

În urma încărcării paginii Web corespunzătoare fișierului HTML anterior, fereastra browserului va arăta astfel:

După cum vedeți, efectele nu sunt spectaculoase în acest caz, dar acesta este unul dintre cele mai simple applet-uri care pot fi create. Teoretic, folosind applet-uri Java, ați putea introduce în paginile dumneavoastră orice doriți (jocuri, aplicații complexe etc.). Trebuie totuși să fiți atenți la dimensiunea fișierului `.class` deoarece acesta

este încărcat în întregime înainte de a se începe executarea sa. Aceste fișiere au dimensiuni relativ mici. De exemplu fișierul **HelloWorld.class** obținut în urma compilării fișierului **HelloWorld.java** folosind compilatorul Java inclus în pachetul de programe *JDK 1.1.6 (Java Development Kit)* are numai 417 bytes.

Între opening tag-ul și closing tag-ul comenzii **<APPLET>**, respectiv **<OBJECT>**, pot fi specificați anumiți parametri care vor fi transmiși applet-ului. Acest lucru poate fi realizat folosind tag-ul **<PARAM>** care are două atribute **NAME** și **VALUE** care vor indica numele, respectiv valoarea parametrului.

Să presupunem că dorim să construim un applet care are un parametru type și care în funcție de valoarea acestui parametru, va afișa cele mai importante date, respectiv cei mai importanți ani din istoria modernă a României. Codul sursă al applet-ului Java este următorul:

```
import java.applet.*;
import java.awt.*;

public class Istorie extends Applet {
  int type;

  private void param() {
    try {type=Integer.parseInt
                  (getParameter("type"));}
    catch (Exception e)
      {showStatus("Parametrul lipseste"+
           "sau este incorect specificat");}
  }

  public void init() {
    param();
  }

  public void paint(Graphics g) {
    if (type==1)
    g.drawString("24 Ianuarie 1859",10,10);
    else g.drawString("1859",10,10);
    g.drawString("Unirea Principatelor",
                                  200,10);
    if (type==1)
    g.drawString("10 Mai 1866",10,30);
    else g.drawString("1866",10,30);
    g.drawString("Incoronarea principelui "
                    +"Carol I", 200, 30);
    if (type==1)
    g.drawString("10 Mai 1877",10,50);
    else g.drawString("1877",10,50);
    g.drawString("Proclamarea Independentei"
           +" de Stat a Romaniei",200,50);
    if (type==1)
    g.drawString("10 Mai 1881",10,70);
    else g.drawString("1881",10,70);
g.drawString("Proclamarea Regatului",200,70);
    if (type==1)
    g.drawString("1 Decembrie 1918",10,90);
    else g.drawString("1918",10,90);
   g.drawString("Desavarsirea Romaniei Mari",
                                  200,90);
    if (type==1)
  g.drawString("30 Decembrie 1947",10, 110);
    else g.drawString("1947",10,110);
    g.drawString("Proclamarea Republicii",
                                  200,110);
    if (type==1)
    g.drawString("22 Decembrie 1989",10,130);
    else g.drawString("1989", 10, 130);
    g.drawString("Inlaturarea regimului"+
                        "comunist",200,130);
  }
}
```

Cei care nu înțeleg aproape nimic nu ar trebui să se "sperie". Nu este nevoie să cunoașteți limbajul Java pentru a putea folosi applet-uri în paginile Web. Există numeroase applet-uri Java utile și interesante, disponibile gratuit pe Internet. Câteva adrese de unde puteți prelua astfel de microaplicații sunt:

Pagini Web cu applet-uri Java

- http://www.javasoft.com/applets/js-applets.html
- http://java.sun.com/applets/js-applets.html
- http://www.gamelan.com/
- http://www.javaworld.com
- http://www.mcp.com/sams/books/235-8/hotlist.htm

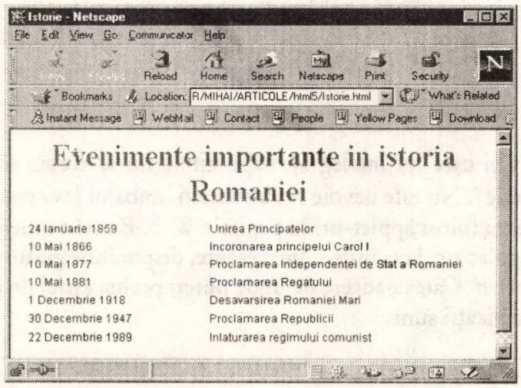
Să studiem comportamentul applet-ului prezentat anterior atunci când primește diferite valori la fel ca pentru parametrul type.

Să considerăm pentru început, următorul fișier HTML:

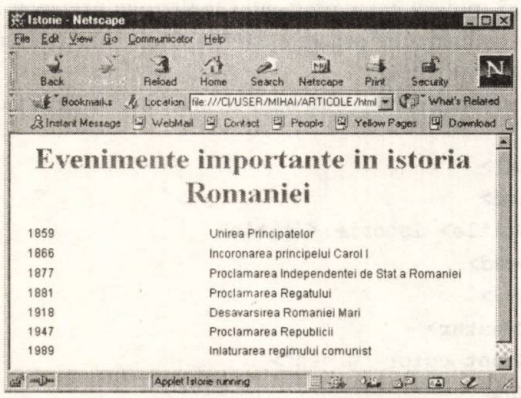
```
<html>
<head>
  <title> Istorie </title>
</head>
<body>
  <center>
  <font color="green">
  <h1>
    Evenimente importante in istoria Romaniei
  </h1>
  </font>
  <object codetype="application/octet-stream"
    classid="java:Istorie.class"
    width = 500 height = 150>
```

```
    <param name = "type" value = "1">
    </object>
    </center>
</body>
</html>
```

După cum vedeți applet-ul este apelat cu valoarea 1 pentru parametrul type. (Chiar și cei care nu știu să programeze în Java își dau seama că aceasta este o valoare particulară pentru respectivul applet, deoarece în textul sursă apare de foarte multe ori condiția if(type==1).) Următoarea imagine prezintă fereastra browserului după încărcarea paginii Web corespunzătoare fișierului HTML prezentat anterior (este evident încărcat și applet-ul referit):



Dacă modificăm valoarea transmisă pentru parametrul type (de exemplu 2457) atunci în momentul executării applet-ului condiția nu va mai fi îndeplinită și efectele vor fi diferite. Următoarea imagine prezintă fereastra browserului pentru cazul în care parametrul type primește valoarea 2457. Nu vom prezenta sursa modificată a fișierului HTML, deoarece singura diferență este înlocuirea numărului 1 cu 2457.



Mai trebuie precizat faptul că fișierul .class nu trebuie neapărat generat de un compilator Java, acest fișier putând fi generat de orice alt compilator (de exemplu un compilator C++) cu condiția ca fișierul rezultat să respecte formatul pe care îl înțelege browserul.

ActiveX

Așa cum a fost introdus VBScript ca alternativă pentru JavaScript, firma Microsoft introduce o alternativă și pentru Java și anume ActiveX, o tehnologie bazată pe *OLE* (*Object Linking and Embedding - Linkeditarea și Împachetarea Obiectelor*) prin implementare OCX. Dacă propoziția anterioară v-a derutat complet, nu e cazul să vă îngrijorați - e valabil pentru peste 95% dintre cei care folosesc ActiveX. Deși este cel mai important rival al limbajului Java, ActiveX nu este un limbaj de programare, ci un standard pentru scrierea programelor. Aceste programe pot fi scrise în Visual Basic, C++ sau aproape orice alt limbaj, inclusiv Java. Microaplicațiile ActiveX poartă denumirea de ***controale*** ActiveX și pot fi folosite nu numai în cadrul paginilor Web. Ele pot fi incluse în orice program scris într-un limbaj de programare care recunoaște tehnologia ActiveX. Datorită faptului că tehnologia ActiveX a fost creată de Microsoft, așa cum probabil vă așteptați, nu este suportată de Netscape Navigator. Totuși un plug-in care rezolvă această problemă este disponibil la adresa http://www.ncompasslabs.com.

Un control ActiveX poate fi introdus într-o pagină Web, folosind tag-ul **<OBJECT>**. Atributele care trebuie transmise în cazul introducerii unui astfel de control sunt cel puțin ciudate, după cum se poate vedea în exemplul următor:

```
<html>
<head>
  <title> Control ActiveX </title>
</head>
<body>
  <center>
  <h1>
    Un control ActiveX
  </h1>
  <object classid = "CLSID:05589FA1-C356-11CE-
                          BF01-00AA0055595A"
    ID="cntrl">
  </object>
  </center>
</body>
</html>
```

Iată și fereastra browserului:

Să explicăm acum ce se întâmplă în cadrul tag-ului **<OBJECT>**. Atributul **CLASSID** are ca valoare un număr care identifică în mod unic controlul ActiveX inclus în pagina dumneavoastră. Acest număr magic poate fi găsit în Registry-ul Windows-ului în `HKEY_CLASSES_ROOT/CLSID`. Dacă propoziția anterioară nu are nici un sens pentru dumneavoastră, atunci va trebui să consultați documentația pe care ați primit-o la cumpărarea produsului respectiv sau să cereți această informație de la persoana care a realizat acest control; deoarece în România este puțin probabil ca respectivul produs să fi fost cumpărat legal și cum autorul nu are nici un interes să dea informații suplimentare celor care au achiziționat acest produs pe "căi ilicite", nici una dintre alternativele de mai sus nu este viabilă.

Salvarea v-o poate aduce un program specializat în crearea paginilor Web, care va descoperi singur numărul de identificare al controlului ActiveX. Atributul *ID* este folosit pentru a identifica respectivul control în eventualele script-uri care apar în pagină. Acest atribut poate avea orice valoare atâta timp cât nu există în aceeași pagină două controale ActiveX cu același ID.

Și controalele ActiveX pot avea anumiți parametri care se specifică la fel ca în cazul applet-urilor și anume prin tag-ul **<PARAM>**. Lățimea și înălțimea zonei pe care o va ocupa controlul ActiveX poate fi specificată cu ajutorul atributelor `WIDTH` și `HEIGHT` a tag-ului **<OBJECT>**.

Deși includerea microaplicațiilor ActiveX în paginile Web este foarte ușoară, ceea ce se află în spatele acestora este foarte greu de înțeles. Informații suplimentare despre utilizarea ActiveX se pot găsi pe Internet.

Pagini Web despre ActiveX

- http://www.microsoft.com/sitebuilder
- http://www.zdnet.com/products/activeuser.html
- http://www.activextra.com

Tag-ul <OBJECT>

Am amintit în paginile anterioare despre un nou tag introdus în specificațiile HTML 4.0 și anume tag-ul **<OBJECT>**.

După cum ați văzut, cu ajutorul acestei comenzi pot fi introduse în paginile Web microaplicații Java și ActiveX. De fapt, cu ajutorul său pot fi incluse aproape orice tip de informații caracteristice paginilor Web de la imagini la programe și chiar alte pagini Web. Numărul atributelor tag-ului **<OBJECT>** este foarte mare, dar acest tag solicită, în primul rând atributul **CLASSID** care are diferite semnificații, în funcție de tipul obiectului introdus.

Există posibilitatea ca anumite tipuri de informații să nu poată fi executate, vizualizate sau redate de browser (de exemplu un sistem non-grafic nu va putea afișa o imagine). Pentru aceste situații trebuie specificate alternative care pot fi comunicate browserului prin imbricarea mai multor tag-uri **<OBJECT>**. Iată o secțiune dintr-un fișier HTML în care este folosită această facilitate:

```
<object title = "Gazeta de Informatica"
        classid = "URL.py">
  <object data = "GInfo.mpeg"
                type = "application/mpeg">
    <object src = "GInfo.gif">
      O prezentare GInfo
    </object>
  </object>
</object>
```

Browserul va încerca să execute programul Python (`.py`) dacă este posibil; dacă nu reușește, va încerca să afișeze un film MPEG; dacă nici filmul nu poate fi afișat, se va încerca afișarea imaginii `"GInfo.gif"`, iar dacă nici acest lucru nu este posibil, va fi afișat textul:

```
O prezentare GInfo.
```

Trebuie precizat faptul că orice tag **<IMG>** poate fi înlocuit cu un tag **<OBJECT>**, deoarece acesta din urmă suportă toate atributele primului. Schimbarea este recomandabilă deoarece, așa cum tag-ul **<APPLET>** a dispărut din specificațiile HTML, nici tag-ul **<IMG>** nu va supraviețui prea mult timp. Iată cum poate fi inclusă o imagine folosind tag-ul **<OBJECT>**:

```
<object data="imag.gif" type = "image/gif">
</object>
```

Tag-ul **<OBJECT>** mai poate fi folosit și pentru a include o altă pagină Web în cadrul paginii curente. Pentru a realiza acest lucru, formatul tag-ului este:

```
<object data ="altapagina.html" </object>
```

Mai trebuie precizat faptul că nu tot ceea ce am spus despre tag-ul **<OBJECT>** este adevărat pentru versiunile actuale ale Netscape Navigator și Internet Explorer. Totuși, tot ceea ce s-a prezentat corespunde specificațiilor HTML 4.0 și probabil că cele două firme (Netscape și Microsoft) care produc aceste browsere vor introduce destul de curând suportul pentru toate facilitățile din specificațiile HTML. Pentru mai multe informații despre tag-ul **<OBJECT>** puteți vizita pagina Web

http://www.microsoft.com/devonly.

Mihai Scorțaru este student la Universitatea Tehnică Cluj; poate fi contactat prin e-mail la adresa: scort@licj.soroscj.ro

PROBLEME pentru ÎNCEPĂTORI

Să nu credeți că o problemă de informatică se poate rezolva doar implementând efectiv, într-un limbaj de programare, un algoritm. Puneți creierul "în priză" și rezolvați următoarele probleme de gândire!

P01: Intersecție de mulțimi

Algoritmul de mai jos determină intersecția a două mulțimi. Acestea sunt memorate în tablourile A având N elemente și B având M elemente.

```
Intersecție:
  K:=0
  Pentru I=1,N execută
    J:=1
    Cât timp J≤M și A(I)B(J) execută
      J:=J+1                                    (*)
    Sfârșit Cât timp
    Dacă J≤M
    atunci K:=K+1: C(K):=A(I)
  Sfârșit Pentru
Sfârșit Intersecție.
```

A. Cunoscând valorile N și M (N ≤ M) determinați care sunt datele pentru care algoritmul este cel mai rapid, adică ce proprietăți trebuie să aibă tablourile A și B pentru ca linia notată cu (*) să se execute de *număr minim de ori*? Care este valoarea acestui minim?

B. Cunoscând valorile N și M (N ≤ M) determinați care sunt datele pentru care algoritmul este cel mai lent, adică ce proprietăți trebuie să aibă tablouri A și B pentru ca linia notată cu (*) să se execute de *număr maxim de ori*? Care este valoarea acestui maxim?

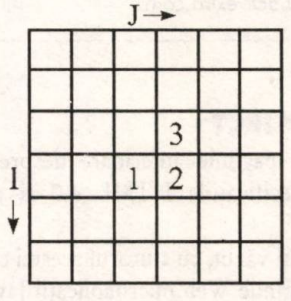
P02: Colorări

Algoritmul de mai jos colorează punctele ecranului pornind din punctul (I,J), în funcție de conținutul tabloului DB.

```
Colorează(I,J):
  K:=1: L:=1
  Cât timp Pe_ecran(I,J) execută
    Desenare_punct(I,J)
    I:=I+T(L,1): J:=J+T(L,2): DB(K):=DB(K)-1
    Dacă DB(K)=0
    atunci K:=K+1: L:=L mod 4 + 1
  Sfârșit Cât timp
Sfârșit Colorează.
```

Subprogramul Pe_ecran(I,J) returnează valoarea true dacă punctul (I,J) este pe ecran (în interiorul lui). Subprogramul Desenare_punct(I,J) luminează punctul (I,J). Conținutul tabloului T este:

T(1,1)=0, T(1,2)=1, T(2,1)=-1, T(2,2)=0,
T(3,1)=0, T(3,2)=-1, T(4,1)=1, T(4,2)=0.



În ce ordine colorează algoritmul punctele aparținând suprafeței de pe figură, dacă tabloul DB are conținutul de mai jos? Copiați figura de 3 ori și completați-o cu următoarele numere de ordine:

A. DB=1,1,2,2,3,3,4,4,5,5,6,...

B. DB=1,2,3,4,5,6,...

C. DB=1,2,2,3,4,4,5,6,...

P03: Reuniune

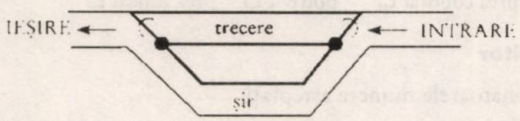
Algoritmul de mai jos construiește din elementele a două șiruri ordonate (A,B) un nou șir ordonat care cuprinde acele elemente care aparțin *cel puțin unuia din cele două șiruri*. Elementele șirului rezultat trebuie să fie *distincte*.

```
Interclasare(A,N,B,M,C,K):
  I:=1: J:=1: K:=0
  Cât timp IN şi JM execută
    K:=K+1
    Dacă A(I)<B(J)
    atunci C(K):=A(I): I:=I+1
    altfel
      Dacă A(I)=B(J)
      atunci C(K):=A(I): I:=I+1: J:=J+1
      altfel C(K):=B(J): J:=J+1
  Sfârşit Cât timp
Sfârşit Interclasare.
```

A. Ce proprietate trebuie să îndeplinească șirurile A și B pentru ca algoritmul să funcționeze corect?

B. Ce eroare se produce în cazul în care aceste proprietăți (condiții) nu sunt îndeplinite?

P04: Coadă specială



Coada specială este o structură de date, pe care se pot executa trei tipuri de operații:

ÎN_COADĂ - pune caracterul sosit de la intrare la sfârșitul cozii;

DIN_COADĂ - scoate caracterul curent din fața cozii și îl pune la ieșire;

TRECE - permite trecerea caracterului sosit la intrare direct la ieșire. (Vezi figura!)

Presupunem că la intrare sosesc caracterele: 1,2,3,4,5.

Exemplu:

Șirul 5,1,2,3,4 se poate obține efectuând operațiile: ÎN_COADĂ, ÎN_COADĂ, ÎN_COADĂ, ÎN_COADĂ, TRECE, DIN_COADĂ, DIN_COADĂ, DIN_COADĂ, DIN_COADĂ.

A. Se pune întrebarea dacă șirurile de mai jos se pot sau nu obține cu operațiile date. Pentru șirurile care nu se pot obține, determinați punctul până la care se poate! Pentru șirurile posibile de obținut, determinați modalitatea de a realiza cerința cu numărul minim de operații necesare!

A1. 3,1,4,2,5 A2. 2,4,3,1,5
A3. 5,1,2,4,3 A4. 2,3,1,5,4

B. Exprimați în cuvinte ordinea posibilă a elementelor din șirurile rezultat!

P05: Actual

Algoritmul de mai jos construiește din elementele a două fișiere *sortate* (A și B), un fișier nou *sortat* (C). Articolele fișierului B sunt comenzi care trebuie efectuate asupra fișierului A. Comenzile pot fi de trei tipuri:

ștergere - articolul din A, având cheia dată, nu se pune în fișierul C!

corecție - nu se va pune în C articolul din A, având cheia dată, ci articolul corespunzător din B!

inserare - se pune în C articolul din B, precizat în comandă!

Structura articolelor din cele două fișiere este:

```
A=Record(cheie: Textul, altceva: Textul)
B=Record(felul: (inserare,ştergere,corecţie),
         cheie: Textul, altceva: Textul)
```

Comenzile din B vor fi efectuate de subprogramul Actualizare:

```
Actualizare(A,B,C):
  Citeşte(A,X): Citeşte(B,Y)
  Cât timp not Eof(A) şi not Eof(B) execută
    Dacă X.cheie < Y.cheie
    atunci Scrie(C,X): Citeşte(A,X)
    altfel
      dacă X.cheie = Y.cheie
      atunci Ceva1
      altfel Ceva2
  Sfârşit Cât timp
Sfârşit Actualizare.

Ceva1:
  Dacă Y.felul = corecţie
  atunci X.altceva:=Y.altceva: Scrie(C,X)
         Citeşte(A,X): Citeşte(B,Y)
  Sfârşit dacă
Sfârşit Ceva1.

Ceva2:
  Scrie(C,(Y.cheie,Y.altceva)): Citeşte(B,Y)
Sfârşit_Ceva2.
```

Presupunem că fișierele A și B sunt valide întotdeauna (au conținuturi care respectă specificațiile problemei): sunt sortate și nu conțin de două ori aceeași cheie. Articolele din fișierul C trebuie de asemenea să conțină chei diferite.

A. Într-un anumit caz acest algoritm funcționează greșit, chiar dacă fișierul B conține doar comenzi corecte.

A1. Care este acest caz?

A2. Ce va fi greșit în rezultat?

A3. Cum se poate corecta greșeala? Argumentați!

B. Ce comenzi greșite (care nu corespund definiției) pot exista în fișierul B?

Vă place GInfo?

Nume: _____ Prenume: _____ Vârsta: _____

Adresa: _____

Localitate: _____ Cod poștal: _____ Județ: _____ Telefon: _____

Ocupația: elev(ă) informatică ☐ elev(ă) ☐ student(ă) ☐ cadru didactic ☐ altă ocupație ☐ (precizați: _____)

Școala (firma, organizația etc.): _____

Dotări

Folosiți un calculator acasă? Da ☐ Nu ☐

Câte calculatoare sunt instalate în școală: (aprox.) _____

Aveți acces la e-mail: Da ☐ Nu ☐

Aveți acces la WWW: Da ☐ Nu ☐

Cum apreciați dotarea cu tehnică de calcul a școlii: _____

Aprecieri generale

Acordați note de la 1 la 10 pe următoarele criterii:

Structura tematică: ___

Calitatea articolelor: ___

Prezentare grafică: ___

Calitate tipografică: ___

Considerați că prețul este:

prea mare ☐ accesibil ☐ prea mic ☐

De cât timp cunoașteți Gazeta de Informatică (ani)? _____

Cum ați procurat acest număr:

chioșc ☐ abonament ☐ cadou ☐ altfel ☐

În acest număr

Apreciați cu note de la 1 la 10 articolele acestui număr:

Efectele negative ale calculatoarelor... ___

Bacalaureat 1999 ___

Calculatoare pe bază de ADN ___

Arhitectura unei baze de date ___

MIRA - limbaj de programare ___

Criptarea simetrică ___

Fire de executare ___

Pagini Web cu HTML 4.0 ___

Probleme pentru începători ___

Articolul care v-a plăcut cel mai mult: _____

De ce? _____

Articolul care v-a plăcut cel mai puțin: _____

De ce? _____

Considerați că ponderea problemelor rezolvate este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că ponderea problemelor propuse este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că ponderea subiectelor teoretice este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că nivelul de accesibilitate al articolelor este

prea coborât ☐ potrivit ☐ prea ridicat ☐

În viitor

În următoarele numere așteptați:

mai multe articole de programare? Da ☐ Nu ☐

mai multe articole teoretice? Da ☐ Nu ☐

mai multe articole despre Internet? Da ☐ Nu ☐

mai multe prezentări de programe? Da ☐ Nu ☐

prezentări de jocuri? Da ☐ Nu ☐

articole despre hardware? Da ☐ Nu ☐

V-ar plăcea să citiți:

pagini redactate de elevii unor licee? Da ☐ Nu ☐

lucrări premiate, realizate de elevi? Da ☐ Nu ☐

actualități școlare? Da ☐ Nu ☐

prezentarea unor situri Web? Da ☐ Nu ☐

tips & tricks? Da ☐ Nu ☐

Comentarii:

_____

_____

_____

_____

_____

_____

Dacă spațiul nu vă este suficient, folosiți coli albe suplimentare

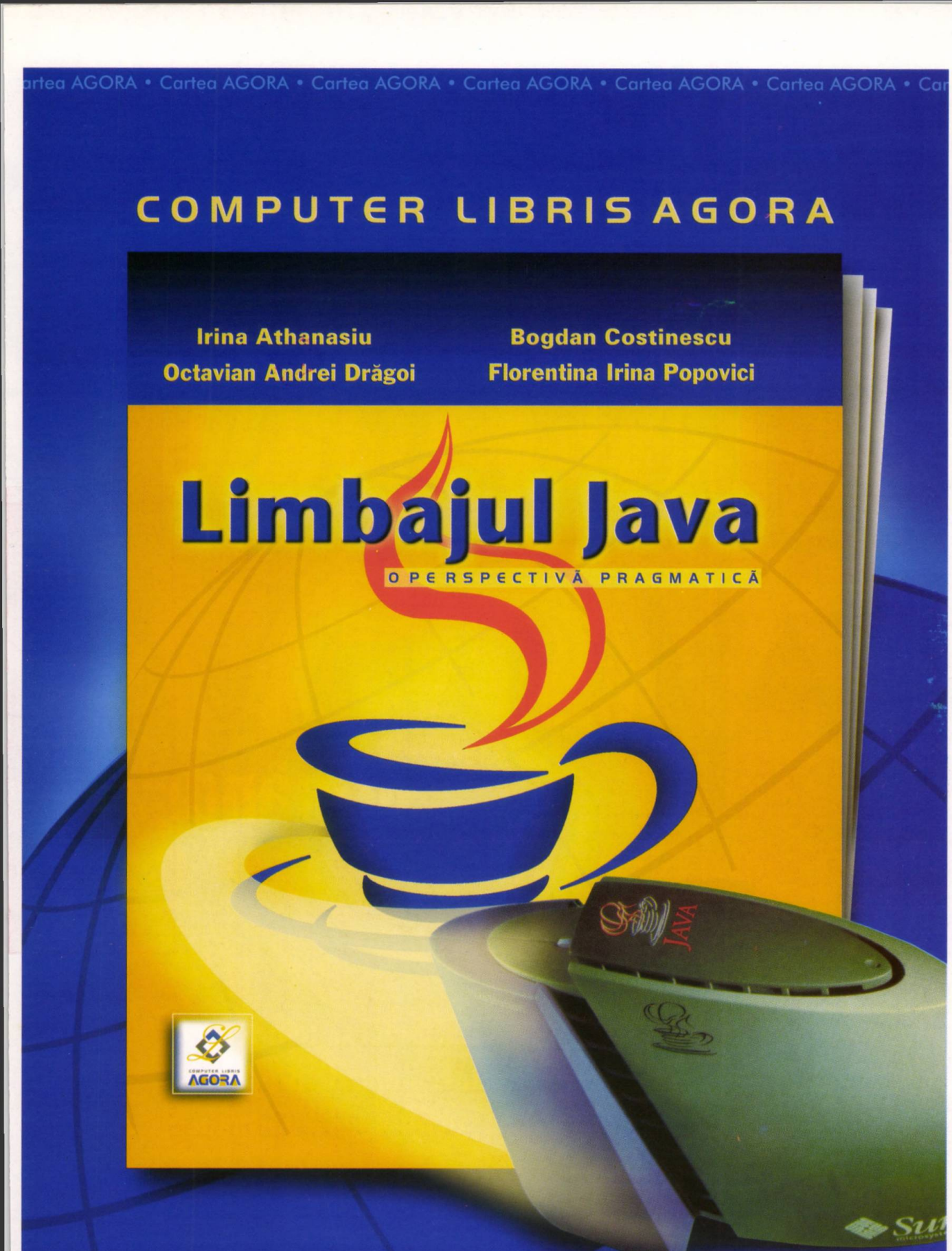
artea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Car

COMPUTER LIBRIS AGORA

Irina Athanasiu
Octavian Andrei Drăgoi
Bogdan Costinescu
Florentina Irina Popovici

Limbajul Java

O PERSPECTIVĂ PRAGMATICĂ



COMPUTER LIBRIS AGORA Cluj-Napoca, Str.Universității Nr.8/8, 3400, Tel. 064-192422
COMPUTER PRESS AGORA Tg.Mureș Str.Tudor Vladimirescu Nr. 63/1 (CP 172/1), Tel. 065-166516, Fax 065-166290
Sucursala București: Str.Poterași, Nr.35/2 (CP 94/49), Tel: 01-3309281, Fax: 01-3309285

Un sfat prinde bine...

PC MAGAZINE ROMANIA

...oricând!

Dacă vrei să te orientezi printre miile de produse și tehnologii informatice de ultimă oră

ABONEAZĂ-TE!

Câștigi **30 000** lei și vei primi cadou și CD-urile **WINSTONE '99** și **WINBENCH '99**

COMPUTER PRESS AGORA, Str.Tudor Vladimirescu Nr.63/1, 4300 Tg.Mureș, Tel.065-166516, Fax 065-166290

Aprilie 1999
Vol. 9/4

Lei 10.000

Gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:

ZERO-polinoame

Soluții la ACM '98

ACTUAL

DirectX

SERIAL

HTML

BEGIN

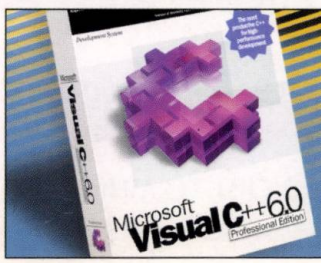
Grafica în 256 culori

INTERNET

Criptare cu chei publice

Plus

Interviu cu echipa care a luat locul I la ACM'98



PREMIUL LUNII

Visual C++ 6.0

COMPUTER PRESS AGORA

PC & BYTE REPORT

TEHNOLOGIA INFORMAȚIEI

ECHIPAMENTE • SISTEME • APLICAȚII • REȚELE • INTERNET

*Abonează-te și plătești doar 120.000 lei (cu 60.000 lei mai puțin decât prețul de vânzare), evitând și ulterioarele majorări de preț

COMPUTER PRESS AGORA • CP.230 OP.1 TG.MUREȘ, Tel. 065-166516 Fax 065-166290

Vol. 9, Nr. 4 1999

gazeta de informatică

Redacția:
Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor:
Mihai Scorțaru (Cluj)
Au mai colaborat:
Victor-Valeriu Patriciu (București)
Doina Rancea (Cluj)
Mihai Oltean (Cluj)
Horea Haitonic (Năsăud)
Sorin Groza (Aiud)

Tehnoredactare:
Dan Crețu, Călin Borz
Coperta și grafica:
Mircea Drăgoi

Adresa pentru corespondență:
C.P. 230-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: (065) 166.516
Fax: (065) 166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *Master Druck*
Târgu-Mureș
tel: 065/162.359
Editor: Computer Press Agora srl
str. T. Vladimirescu nr. 63/1
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Gabriela Bucșa (gbucsa@agora.ro)
Denis Mărioara (mdenis@agora.ro)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: (01) 330.92.81

Σ Sumar

probleme.. PAG. 5

propuse pentru REZOLVARE

Olimpiada Județeană de Informatică - Cluj

soluții.. PAG. 7

probleme REZOLVATE
pagini realizate de Horia GEORGESCU

Soluțiile problemelor date la ACM '98

actual.. PAG. 16

DirectX
Mihai OLTEAN

bac.. PAG. 20

Bacalaureat 1999
Doina RANCEA

Rezolvarea subiectului publicat în GInfo 3

focus.. PAG. 26

Zero-polinoame
Mihai SCORȚARU

internet.. PAG. 32

Criptarea simetrică
Victor-Valeriu PATRICIU

serial.. PAG. 36

Pagini WEB cu HTML 4.0
Mihai SCORȚARU

begin.. PAG. 48

Grafică în 256 de culori
Horea HAITONIC

Din secretele șirurilor
Sorin GROZA

interviu.. PAG. 23

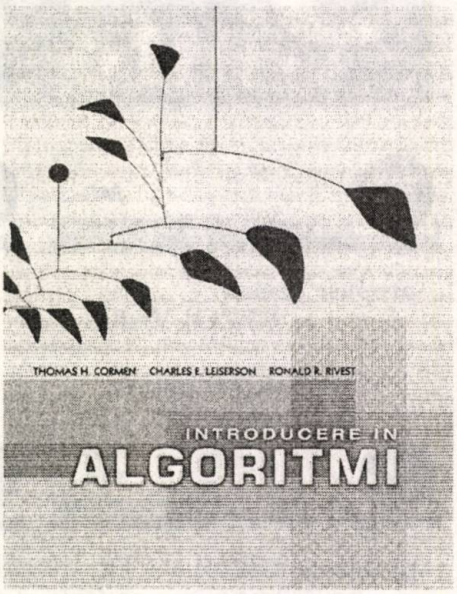
Păreri și păreri...
Horia GEORGESCU

ale componenților echipei care a câștigat locul I la concursul ACM '98

COMPUTER LIBRIS AGORA

În curând o nouă apariție editorială

La începutul lunii mai va apare în sfârșit și în limba română celebra carte de ALGORITMI scrisă de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, publicată de MIT Press, SUA. Este o carte care de la prima ei apariție a fost retipărită, numai în Statele Unite, de 21 ori.



Această carte oferă o introducere exhaustivă a studiului modern al algoritmilor. Analiza și proiectarea algoritmilor tratați este accesibilă și asigură condițiile unui studiu, pe de o parte fundamental, pe de altă parte profund.

Fiecare capitol prezintă cel puțin un algoritm sau o tehnică de proiectare, un domeniu de aplicații, respectiv un subiect înrudit. Algoritmii sunt descriși în limba română și în pseudocod. Cartea conține peste 260 de figuri care ilustrează modul de funcționare al algoritmilor. Deoarece accentul este pus pe eficiență ca și criteriu de proiectare, include analize atente ale timpului de executare a tuturor algoritmilor.

Cartea originală conține 1000 pagini și este structurată în șapte părți, subdivizate în 37 capitole, cu o bibliografie de 205 titluri și o listă bibliografică de 30 pagini.

Partea I, "Fundamente matematice", este un compendiu al metodelor și instrumentelor care vor fi utilizate în analiza algoritmilor. Este organizată esențial pentru referință, ca un fel de tutorial.

Partea a II-a, "Sortare și statistici de ordine", tratează problematica tehnicilor de sortare. Sunt studiate metodele *heapsort, quicksort și tehnici de sortare în timp liniar, precum și statisticile de ordine.*

Partea a III-a, "Structuri de date", prezintă unele tehnici de bază pentru reprezentarea și manipularea mulțimilor dinamice finite. Sunt discutate *structuri de date elementare, tabele de dispersare, arbori de căutare binară, arbori roșu-negru, extinderea structurilor de date.*

Partea a IV-a, "Tehnici avansate de proiectare și implementare", acoperă trei tehnici importante pentru proiectarea și analiza algoritmilor eficienți: *programarea dinamică, algoritmi greedy și analiza amortizată.* Aceste tehnici sunt ceva mai sofisticate decat altele, dar sunt esențiale pentru faptul că permit abordarea efectivă a multor probleme computaționale.

Partea a V-a, "Structuri de date avansate", revine la examinarea structurilor de date care suportă operații pe mulțimi dinamice, dar la un nivel mai avansat decat cel din Partea a III-a. Sunt discutați *B-arbori, ansambluri binomiale, ansambluri Fibonacci, precum și structuri de date pentru mulțimi disjuncte.*

Partea a VI-a, "Algoritmi de grafuri", tratează unele dintre cele mai importante probleme computaționale definite pe grafuri. Sunt trecuți în revistă algoritmi elementari pe grafuri, arbori de acoperire minimă, problematica celor mai scurte drumuri și algoritmi pentru determinarea fluxului maxim.

Partea a VII-a, "Subiecte diverse", conține o selecție de subiecte care extinde și completează materialul din restul cărții. Unele capitole introduc modele de calul noi precum circuite combinatoriale sau computere paralele. Altele acoperă domenii mai specializate precum geometria computațională sau teoria numerelor. Ultimele două capitole discută despre unele dintre limitările cunoscute ale proiectării algoritmilor eficienți și introduc tehnici pentru studierea acestor limitări.

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422]

Ce fel de om ești, PROFESORE?

Dezbatem cu ardoare și interes programe școlare, rezultate obținute la concursuri, prezentul și viitorul învățământului în general și pe discipline, dar se pare că uităm ceva esențial. Școala este populată de elevi și profesori, și parafrazând o sintagmă bine cunoscută mă întreb: ***elevii sunt în școală ca profesorii să aibă cui să predea, sau profesorii sunt acolo pentru că au o menire, și anume trebuie să educe și să instruiască elevii?!***

Întrebarea este, evident, retorică, și aparent subiectul s-a și încheiat. Dar,... nu aș fi început această incursiune dacă nu doream să vă povestesc câteva întâmplări care m-au pus pe gânduri.

Petrică

... a fost elev acum mulți ani. Când era în clasa a VI-a, i s-a întâmplat ceva deosebit de trist: după luni de suferință mama lui a murit. Puternic marcat de aceasta și-a neglijat studiile. A devenit arțăgos, neatent, aparent superficial în tot ceea ce făcea. Profesoara de matematică, nouă în școală, ambițioasă și pretențioasă "l-a abțiguit" de câte ori putea, o chiar irita indiferența afișată de Petrică, dar nu a încercat niciodată să afle motivele. De fapt Petrică iubea matematica, iar doamna profesoară semăna foarte mult cu mama lui. La sfârșit de an a venit și el la școală cu un buchet de flori, ce-i drept mai ieftine, flori pe care i le-a oferit cu drag profesoarei... Din păcate, a doua zi, și-a revăzut florile în coșul de gunoi.

Efectul? Nu a mai învățat la matematică decât strictul necesar, cu o veșnică amărăciune și de data aceasta cu o reală indiferență.

Măriuca

... a urmat gimnaziul într-un sat, relativ aproape de un oraș mare. Se apropia ziua de 1 martie și în acel sat nu se vindeau mărțișoare. Dar Măriuca l-a rugat pe Bunicul să cumpere un mărțișor de la oraș ca să-l poată oferi Doamnei diriginte, care "este atât de frumoasă...". Măriuca nu știe nici azi că doamna dirigintă a aruncat disprețuitor mărțișorul pe masa cancelariei, comentând: "Ăsta-i mărțișor?" În mintea Măriucăi a rămas răceala doamnei diriginte, faptul că aceasta nici măcar nu i-a zâmbit în semn de mulțumire, dimpotrivă, de atunci e privită cu dispreț.

Valer

... a fost student și trebuia să efectueze așa numita "practică pedagogică". A fost plin de incertitudini, avea emoție numai la gândul că va trebui să iasă în fața unei clase de liceeni care sunt cu doar câțiva ani mai mici ca el, să le explice lecția, să le corecteze eventualele greșeli și să le răspundă la întrebări. Inițial a frecventat orele ținute de profesoara la catedra căreia a fost repartizat, cu deosebită curiozitate, dornic să învețe, să "copieze" tot ceea ce poate fi folositor. Dar profesoara povestea în timpul orei cu o altă profesoară, sau enunța o problemă, apoi lăsa copiii să facă ce doresc. În plus, de multe ori nici Valer nu înțelegea de ce "ploua" cu nota 4!?

Efectul? Valer nu a mai dorit să fie profesor...

Andrei

... a fost student acum câțiva ani. Într-o zi a ieșit la tablă să rezolve o problemă. În timp ce profesorul răspundea unuia dintre studenți, Andrei a început să scrie rezolvarea pe tablă. Profesorul, mirat, l-a întrebat: "Dumneata ce faci?" Andrei a încercat să se explice: "M-am gândit că...", dar a fost întrerupt: "Dumneata nu gândești! Dumneata scrii pe tablă ceea ce îți dictez eu!"

Efectul? Dacă Andrei a mai "gândit" vreodată rezolvarea unei probleme, a ținut-o secret...

Mihai

... a fost un copil ca oricare altul, dar a avut un noroc teribil. Profesorul lui de matematică iubea copiii, iubea școala, iubea matematica.

Efectul? În clasa a V-a 28 de copii au declarat la o măsurătoare sociometrică că disciplina pe care o iubesc cel mai mult este... matematica!

În loc de concluzii

Nu este cazul să trag morala din aceste "fabule" copilărești. Nu este cazul să generalizez, crezând că povestioarele de mai sus reprezintă ceva semnificativ în ceea ce privește învățământul! Cred că fiecare dintre ele are și o poveste pereche în care profesorul sau profesoara au reacționat contrar celor de mai sus și "fratele" lui Petrică este acum, la rândul lui profesor de matematică care își aduce aminte cu plăcere de anii petrecuți în școală. "Sora" Măriucăi este frumoasă, se îmbracă cu mult gust, pentru că diriginta ei a învățat-o cum să se comporte, ce este frumos și că adevărata frumusețe radiază din suflet. Valer, văzând că e nevoie de profesori care înțeleg valoarea meseriei, a devenit totuși profesor și nu va uita niciodată cum să "nu procedeze". Iar Andrei? Andrei este programator la Microsoft.

Clara Ionescu

Descoperiți modul de lucru Visual

Acum vă puteți permite(*) un instrument de lucru original

(*) peste 70% reducere
numai pentru persoane fizice
numai pentru uz personal, acasă

Oferta este valabilă în perioada 01.04 - 30.06 1999

Denumirea produsului	Preț (USD)
VB Std 6.0 Win 32 English Intl CD Lrng	31
VB Pro 6.0 Win 32 Romanian CVUP CD	71
VB Pro 6.0 Win 32 Romanian CD	101
VC++ Std 6.0 Win 32 English Intl CD	31
VC++ Pro 6.0 Win 32 English Intl CVUP CD	71
VC++ Pro 6.0 Win 32 English Intl CD	101

Denumirea produsului	Preț (USD)
VFoxPro Pro 6.0 Win 32 English Intl CVUP CD	71
VFoxPro Pro 6.0 Win 32 English Intl CD	101
VInterDEV 6.0 Win 32 English Intl CVUP CD Pro	71
VInterDEV 6.0 Win 32 English Intl CD Pro	101
VJ++ Std 6.0 Win 32 English Intl CD	31
VJ++ Pro 6.0 Win 32 English Intl CVUP CD	71
VJ++ Pro 6.0 Win 32 English Intl CD	101

Prețul include 70% discount și se plătește în lei, la cursul de schimb al B.N.R. Prețul nu include T.V.A.

Pentru a beneficia de această ofertă,
contactați firma GeCAD, Microsoft Developer Dealer, tel.: 01-324 84 09,
http://www.gecad.ro/mspromo

PERSPECTIVA 2000

Microsoft®

Încercați-vă puterile!

PROBLEME propuse pentru REZOLVARE

În toiul olimpiadelor de informatică este firesc să vă oferim câteva din problemele etapei județene. În rest, vă dorim succes la Olimpiada Națională!

P049901: Premii

La un concurs doi concurenți au obținut cel mai bun rezultat. Premiul oferit de sponsori este format din mai multe obiecte, având valori nu neapărat distincte.

Scrieți un program care împarte obiectele între cei doi concurenți, astfel încât diferența dintre valorile totale ale obiectelor primite de fiecare să fie minimă.

Date de intrare

Datele de intrare se vor citi din fișierul text **PREMII.IN**:
- pe prima linie a fișierului se află un număr întreg n ($1<n\leq100$) reprezentând numărul de obiecte;
- pe a doua linie sunt scrise valorile celor n obiecte; acestea sunt numere întregi, strict pozitive, mai mici decât 100, separate printr-un singur spațiu.

Date de ieșire

Fie S1 și S2 sumele valorilor obiectelor primite de cei doi concurenți. În fișierul **PREMII.OUT** se vor scrie numerele S1 și S2, în orice ordine, separate printr-un singur spațiu.

Exemplu

PREMII.IN	PREMII.OUT
5	85 90
10 80 30 50 5	

P049902: Numere "superprime"

Numim numere *superprime* acele numere naturale prime ale căror prefixe sunt de asemenea prime. De exemplu, numărul 239 este superprim, deoarece numerele 2, 23 și 239 toate sunt prime. Numărul 241 nu este superprim, deoarece 24 nu este prim.

Scrieți un program care generează toate numerele superprime formate din n cifre.

Date de intrare

Din fișierul text **PRIME.IN** se va citi numărul cifrelor numerelor prime care trebuie generate ($1<n\leq9$).

Date de ieșire

În fișierul **PRIME.OUT** se vor scrie numerele prime generate, precedate de prefixele lor prime. Pe fiecare linie a fișierului se va scrie un număr format din n cifre, precedat de prefixele lui. Două numere se vor despărți printr-un singur spațiu. Fișierul va avea atâtea linii câte numere superprime a găsit algoritmul. În cazul în care nu există nici un număr superprim, în fișierul de ieșire se va scrie: Nu

Exemplu

Datele de ieșire sunt scrise pe două coloane din motive de spațiu.

PRIME.IN	PRIME.OUT	
3	2 23 233	3 37 379
	2 23 239	5 59 593
	2 29 293	5 59 599
	3 31 311	7 71 719
	3 31 313	7 73 733
	3 31 317	7 73 739
	3 37 373	7 79 797

P049903: Panou

Se consideră un panou electric pe care sunt dispuse n*m becuri, conform unui caroiaj. Aceste becuri pot fi aprinse sau stinse. Schimbarea stărilor se poate realiza de la un pupitru de comandă care are la dispoziție un mecanism de dimensiune p*q, care așezat deasupra panoului îl acoperă pe acesta parțial și schimbă starea becurilor pe care le acoperă.

Scrieți un program care stabilește dacă se pot aprinde, sau nu, toate becurile de pe panou, folosind acest dispozitiv.

Date de intrare

Datele de intrare se vor citi din fișierul text **PANOU.IN**:
- pe prima linie a fișierului se află numerele naturale n și m ($1<n,m\leq100$) reprezentând numărul de linii, respectiv numărul de coloane ale panoului electric;
- a doua linie conține numerele naturale p și q, reprezentând numărul de linii, respectiv de coloane ale dispozitivului ($1\leq p,q\leq100$, $p\leq n$, $q\leq m$);

- pe fiecare din următoarele n linii sunt scrise stările celor m becuri de pe acea linie; fiecărui bec aprins îi corespunde un caracter +, iar celor stinse câte un caracter - (caracterele + și - nu sunt separate prin nici un spațiu).

Date de ieșire

Dacă toate becurile se pot stinge, pe prima linie a fișierului **PANOU.OUT** se va scrie cuvântul DA, altfel se va scrie NU. În caz afirmativ, pe următoarele linii se vor descrie pozițiile dispozitivului care vor avea ca efect stingerea tuturor becurilor. O poziție se precizează prin indicele de linie și cel de coloană din matricea corespunzătoare panoului electric, unde se poziționează colțul stânga sus al dispozitivului.

Exemplu

PANOU.IN	PANOU.OUT
5 6	DA
2 2	2 3
++++++	3 3
++--++	
++++++	
++--++	
++++++	

P049904: Sume

Se dă un șir strict crescător de n numere naturale.

Să se decidă dacă orice număr natural $x \in \{1, \ldots, m\}$ se poate scrie ca sumă de termeni distincți din șirul dat ($1<n\leq 100.000$, $m\leq 2.000.000.000$). Dacă descompunerea nu este posibilă, să se determine cel mai mic număr care nu poate fi scris ca sumă de termeni distincți din șirul dat.

Date de intrare

Datele de intrare se vor citi din fișierul text **SUME.IN**. Pe prima linie a fișierului sunt scrise numerele n și m separate printr-un singur spațiu. Pe următoarele n linii sunt scrise elementele șirului strict crescător.

Date de ieșire

Pe prima linie a fișierului **SUME.OUT** se va scrie DA, în cazul în care descompunerea este posibilă; în caz contrar, se va scrie NU, iar pe următoarea linie va fi tipărit cel mai mic număr care nu poate fi scris ca sumă de termeni distincți din șirul dat.

Exemple

SUME.IN	SUME.OUT	SUME.IN	SUME.OUT
4 11	DA	4 11	NU
1		1	8
2		2	
4		4	
5		10	

P049905: Dominouri

Se dau n piese de domino (o piesă de domino este o plăcuță dreptunghiulară pe care sunt înscrise două cifre din mulțimea $\{0,1,\ldots,6\}$). Cu aceste piese se pot forma lanțuri (formațiuni liniare) în care oricare două dominouri consecutive sunt alipite pe zona unde cifrele lor sunt identice.

Scrieți un program care din cele n dominouri date formează două lanțuri astfel încât diferența dintre suma tuturor punctelor înscrise pe dominourile din prima formație și suma tuturor punctelor înscrise pe dominourile din cea de a doua formație să fie minimă; dominourile pot fi rotite. *În plus, cele două lanțuri formate trebuie să fie subșiruri ale șirului inițial de dominouri.*

Date de intrare

Datele de intrare se vor citi din fișierul text **DOMINO.IN**.

Pe prima linie a fișierului se află numărul dominourilor ($1<n\leq 20$) din care se vor forma cele două lanțuri.

Pe următoarele n linii se află valorile scrise pe cele n dominouri; acestea sunt perechi de numere naturale aparținând mulțimii $\{0, \ldots, 6\}$. Între cele două numere există un singur spațiu.

Restricții:

1. Trebuie utilizate toate piesele de domino; datele de intrare permit formarea a două lanțuri.
2. Cele două lanțuri formate trebuie să fie subșiruri ale șirului inițial de dominouri (nici un domino nu va depăși în propria formație un alt domino care în șirul din fișierul de intrare a fost precizat undeva, în urma acestuia).

Date de ieșire

Datele de ieșire se vor scrie în fișierul **DOMINO.OUT**, astfel:
- pe prima linie numărul dominourilor din primul lanț;
- pe următoarele linii dominourile din primul lanț;
- pe următoarea linie numărul dominourilor din al doilea;
- pe următoarele linii dominourile din al doilea lanț.

Exemplu

DOMINO.IN	DOMINO.OUT
5	3
1 0	0 1
1 5	1 5
2 3	5 1
2 5	2
5 1	3 2
	2 5

P049906: Factorial

Din fișierul **FACT.IN** se citește un număr natural N ($1.000\leq N\leq 10^{1.000}$) despre care se știe că reprezintă o valoare **k!** (**k** factorial). Scrieți un program care determină și afișează în fișierul **FACT.OUT** numărul **k**.

Exemple

FACT.IN	FACT.OUT
40320	8

FACT.IN	FACT.OUT
1307674368000	15

SOLUȚIILE problemelor propuse

ACM, chiar dacă este un concurs organizat pentru studenți, diferă de concursurile destinate elevilor de liceu prin faptul că se concurează pe echipe, iar evaluarea are loc în "timp real". Sperăm că soluțiile prezentate vă vor fi utile în scopul pregătirii pentru olimpiade. Soluțiile publicate mai jos aparțin echipei câștigătoare a Universității București. Puteți cunoaște mai bine membrii acestei echipe citind interviul realizat cu ei în acest număr al revistei.

P029901: Țara Confuziei

Harta Țării Confuziei seamănă foarte mult cu un *automat finit nedeterminist* (un graf orientat în care muchiile sunt etichetate). Dacă se încearcă găsirea soluției prin metoda backtracking, complexitatea este exponențială.

Codificăm `N` cu 1, `E` cu 2, `S` cu 3 și `V` cu 4. În matricea `a` păstrăm harta astfel: `a[i,j,0]` reprezintă numărul intersecțiilor în care se poate ajunge din intersecția `i`, urmând direcția `j`. Pentru `0<k≤a[i,j,0]`, `a[i,j,k]` reprezintă o intersecție în care se poate ajunge din `i` pe direcția `j`.

Vom folosi un vector `b` având ca dimensiune numărul `max` al intersecțiilor. Vectorul `b` este vectorul caracteristic al submulțimii de intersecții la care se poate ajunge la momentul curent, moment corespunzător numărului de caractere analizate din șirul ce trebuie testat. Deci `b[i]=1` dacă și numai dacă la momentul curent se poate ajunge în intersecția `i`. Inițial (dacă s-au citit 0 caractere) toate componentele lui `b`, cu excepția celei corespunzătoare intersecției de plecare, au valoarea 0.

La citirea unui nou caracter `ch∈{'N','S','E','V'}` din șirul de intrare ce trebuie testat, noul `b` va fi calculat astfel:

```
c[i]←false, ∀ i=1,...,max
for i=1,max do
    if b[i]=1
    then for t=1,a[i,ch,0] do
            c[a[i,ch,t]]←true
b←c
```

Se observă că aceaste calcule se fac în timp polinomial pentru toate intersecțiile, iar în final se verifică dacă pe poziția din vectorul `b`, corespunzătoare intersecției destinație, apare sau nu 1.

Listing: FUZZY.PAS

```
const max = 40;
var a:array[1..max,1..4,0..max] of Byte;
type mat=array[1..max] of Boolean;
var b,c:mat;
    n,start,stop:Byte;
    ntest:Integer;

procedure init;
var i,j,k,t:Byte;
begin
  Readln(start,stop);
  for i:=1 to n do
    for j:=1 to 4 do
    begin
      Read(k);
      if k=0
      then
      begin
        a[i,j,0]:=0; Readln
      end
      else begin
             t:=1;
             a[i,j,t]:=k;
             while not SeekEoln do
             begin
               Read(k);
               Inc(t);
               a[i,j,t]:=k
             end;
             Readln; a[i,j,0]:=t
           end
    end
end;
```

```
function doit(s:string):Boolean;
var i,j,t:Byte;
begin
  for i:=1 to max do
    b[i]:=false;
  b[start]:=true;
  while s<>'' do
  begin
    j:=0;
    if s[1]='N' then j:=1;
    if s[1]='E' then j:=2;
    if s[1]='S' then j:=3;
    if s[1]='W' then j:=4;
    Delete(s,1,1);
    for i:=1 to max do
      c[i]:=false;
    if j=0 then Continue;
    for i:=1 to n do
      if b[i] then
        for t:=1 to a[i,j,0] do
          c[a[i,j,t]]:=true;
    b:=c
  end;
  doit:=b[stop]
end;

procedure process;
var s:string;
    i,t:Byte;
begin
  Readln(t);
  for i:=1 to t do
  begin
    Readln(s);
    if doit(s)
    then Writeln('YES')
    else Writeln('NO')
  end;
  Readln
end;

Begin
  Assign(input,'A.DAT'); Reset(input);
  Readln(n);
  ntest:=1;
  while n<>0 do
  begin
    init;
    Writeln('Fuzzylad #',ntest,':');
    process;
    Writeln;
    Readln(n);
    Inc(ntest)
  end;
  Close(input)
End.
```

P029902: Numărul majoritar

Problema a fost prezentată în Gazeta de Informatică nr. 1/1999, în articolul "Alegeri prezidențiale". Va fi prezentată mai jos o altă metodă de rezolvare.

Cel mai simplu algoritm liniar presupune să folosim un vector indexat după toate numerele posibile, în care să memorăm pentru fiecare număr frecvența aparițiilor sale.

Deoarece un asemenea vector ar ocupa 16 GB, folosim un artificiu ce constă în a considera pentru fiecare număr cei 4 octeți din reprezentarea sa.

Folosim o matrice `frecventa[0..3,0..255]`, inițializată cu 0, cu următoarea semnificație:
`frecventa[i,j]`= de câte ori octetul `i` din numerele din șirul de intrare este egal cu `j`.

Afirmăm că dacă există un număr majoritar în șir, atunci el este compus din unul din octeții ce se repetă cel mai des pe poziția `0` (fie el `majoritar[0]`) plus unul din octeții ce se repetă cel mai des pe poziția `1` (fie el `majoritar[1]`) etc. Afirmația are la bază faptul că dacă există un număr majoritar, atunci el este unic.

Într-adevăr, dacă alegem un octet `o[i]` care nu se repetă cel mai des, el nu poate face parte dintr-un număr majoritar. De asemenea, dacă pentru un `i=0,1,2,3` există mai mulți octeți care se repetă cel mai des, nu are importanță pe care îl alegem, deoarece nu va exista un număr majoritar. De aceea vom alege pentru fiecare `i=0,...,` un octet majoritar pe linia `i` a matricei `frecventa` și vom forma din ei un număr candidat la a fi majoritar. Rămâne să verificăm dacă acest număr este într-adevăr majoritar, ceea ce se realizează verificând dacă el apare de mai mult de `n/2` ori în șirul de intrare.

Observație:
Pentru a citi o singură dată din fișierul de intrare, împreună cu calculul matricei `frecventa`, facem o copie a acestui fișier pe care o folosim la a doua parcurgere și pe care apoi o "distrugem".

Listing: NUMAR.PAS

```
var ftmp:Text;
    numar,i,j,n,candidat,prezent:Longint;
    frecventa:array[0..3,0..255] of Longint;
    majoritar:array[0..3] of Byte;

Begin
  Assign(input,'B.DAT'); Reset(input);
  Assign(ftmp,'TMP.UB1');
  while not SeekEof do
  begin
    Read(n);
    Fillchar(frecventa,Sizeof(frecventa),0);
    Rewrite(ftmp);
    for i:=1 to n do
    begin
      Read(numar);
```

```
      for j:=0 to 3 do
        Inc(frecventa[j,(numar shr (8*j))
                                  and 255]);
      Writeln(ftmp,numar)
    end;
    candidat:=0;
    for j:=3 downto 0 do
    begin
      majoritar[j]:=0;
      for i:=1 to 255 do
        if frecventa[j,majoritar[j]]<
                          frecventa[j,i]
        then majoritar[j]:=i;
      candidat:=(candidat shl 8) or
                            majoritar[j]
    end;
    prezent:=0;
    Reset(ftmp);
    for i:=1 to n do
    begin
      Read(ftmp,numar);
      if numar=candidat then Inc(prezent)
    end;
    Close(ftmp); Erase(ftmp);
    if prezent>n/2
    then Writeln(candidat)
    else Writeln('Nu exista numar majoritar')
  end;
  Close(input)
End.
```

P029903: Rețele sigure

Definiția unei *subrețele maximale* este de fapt definiția unei *componente biconexe a grafului de calculatoare.*

Folosim algoritmul cunoscut de determinare a componentelor biconexe ale unui graf neorientat; complexitatea în timp a algoritmului este de ordinul $O(n*m)$, unde n este numărul de vârfuri și m numărul de muchii (vezi Gazeta de Informatică nr.2/1996). În plus, ordonăm calculatoarele în fiecare subrețea și ordonăm subrețelele după primul calculator.

Listing: RETELE.PAS

```
{$M 65520,0,655360}
const max=6400;
type lmuchie=^muchie;
     muchie=record v1,v2:Integer;
                   u:lmuchie
            end;
     vecint=array[0..max] of Integer;
var adiac,comp:array[0..max] of ^vecint;
    n,ncomp:Integer;
procedure parcurge;
var ndf:Integer;
    s,s2:lmuchie;
    tata,nrdf,v:vecint;
  procedure bloc(i:Integer);
  var j:Integer;
    procedure componenta;
    var k:Integer;
        prezenti,pointeri:vecint;
      procedure baga(x:Integer);
      begin
        if (pointeri[x]<1) or
           (pointeri[x]>prezenti[0]) or
           (prezenti[pointeri[x]]<>x)
        then begin
               Inc(prezenti[0]);
               prezenti[prezenti[0]]:=x;
               pointeri[x]:=prezenti[0]
             end;
      end;

    begin
      prezenti[0]:=0; s2:=s;
      repeat
        baga(s2^.v1); baga(s2^.v2);
        if (s2^.v1=i) and (s2^.v2=j)
        then Break;
        s2:=s2^.u;
      until false;
      s:=s2^.u; Inc(ncomp);
      getmem(comp[ncomp],(prezenti[0]+1)*
                         Sizeof(Integer));
      Move(prezenti,comp[ncomp]^,
           (prezenti[0]+1)*Sizeof(Integer))
    end;

  var k:Integer;
  begin
    Inc(ndf); nrdf[i]:=ndf; v[i]:=ndf;
    for k:=1 to adiac[i]^[0] do
    begin
      j:=adiac[i]^[k];
      if (nrdf[i]>nrdf[j]) and (j<>tata[i])
      then begin
             New(s2);
             s2^.v1:=i; s2^.v2:=j;
             s2^.u:=s; s:=s2
           end;
      if nrdf[j]=0
      then begin
             tata[j]:=i; bloc(j);
             if v[i]>v[j] then v[i]:=v[j];
             if v[j]>=nrdf[i]
             then componenta
           end
      else
        if v[i]>nrdf[j]
        then v[i]:=nrdf[j]
    end
  end;
```

```
var i:Integer;
begin
  ncomp:=0; ndf:=0; s:=nil;
  for i:=0 to n-1 do
  begin
    nrdf[i]:=0; v[i]:=0; tata[i]:=-1
  end;
  for i:=0 to n-1 do
  begin
    if nrdf[i]=0 then bloc(i);
    if adiac[i]^[0]=0
    then begin
           Inc(ncomp);
           Getmem(comp[ncomp],2*
                          Sizeof(Integer));
           comp[ncomp]^[0]:=1;
           comp[ncomp]^[1]:=i
         end
  end
end;

procedure afiseaza;
var perm:array[1..max] of Integer;
  procedure qsortcomp(l,r:Integer);
  var i,j,x,y:integer;
  begin
    if l<r
    then
    begin
      i:=l; j:=r;
      x:=comp[ncomp]^[l+random(r-l+1)];
      repeat
        while comp[ncomp]^[i]<x do Inc(i);
        while x<comp[ncomp]^[j] do Dec(j);
        if i<=j
        then
        begin
          y:=comp[ncomp]^[i];
          comp[ncomp]^[i]:=comp[ncomp]^[j];
          comp[ncomp]^[j]:=y;
          Inc(i); Dec(j)
        end
      until j<i;
      qsortcomp(l,j);
      qsortcomp(i,r)
    end
  end;

  procedure qsortperm(l,r:Integer);
  var i,j,x,y:Integer;
  begin
    if l<r
    then
    begin
      i:=l; j:=r;
      x:=comp[perm[l+random(r-l+1)]]^[1];
      repeat
        while comp[perm[i]]^[1]<x do Inc(i);
        while x<comp[perm[j]]^[1] do Dec(j);
        if i<=j
        then
        begin
          y:=perm[i];
          perm[i]:=perm[j];
          perm[j]:=y;
          Inc(i); Dec(j)
        end;
      until j<i;
      qsortperm(l,j);
      qsortperm(i,r)
    end;
  end;
var i,j:Integer;
begin
  for ncomp:=1 to ncomp do
  begin
    qsortcomp(1,comp[ncomp]^[0]);
    perm[ncomp]:=ncomp
  end;
  qsortperm(1,ncomp);
  writeln(ncomp,' sub-retele');
  for i:=1 to ncomp do
  begin
    for j:=1 to comp[perm[i]]^[0] do
      Write(comp[perm[i]]^[j],' ');
    Writeln
  end;
  Writeln
end;

var i,j,nr,k,tz:Integer;
    ch:Char;
    s:string;
    h:pointer;

Begin
  Assign(input,'C.DAT'); Reset(input);
  while not SeekEof do
  begin
    Read(n); mark(h);
    for tz:=1 to n do
    begin
      Read(i);
      if SeekEof then;
      Read(ch);
      s:='';
      repeat
        Read(ch);
        if Ch in ['0'..'9']
        then s:=s+ch;
      until not (ch in ['0'..'9']);
      while ch<>')' do Read(ch);
```

```
      Val(s,nr,k);
      Getmem(adiac[i],(1+nr)*Sizeof(Integer));
      adiac[i]^[0]:=nr;
      for j:=1 to nr do Read(adiac[i]^[j])
    end;
    parcurge; afiseaza;
    Release(h)
  end;
  Close(input)
End.
```

P029904: Drumuri pe autostradă

Definim un graf orientat G_1, ale cărui vârfuri corespund punctelor de pe hartă; două vârfuri sunt *adiacente* dacă punctele corespunzătoare sunt puncte libere adiacente. Unul din vârfurile corespunzătoare oraşelor îl notăm cu `start`, iar pe celălalt îl notăm cu `stop`. Definim ***costul unui arc*** `(x,y)` ca fiind:

- 1+costul punctului corespunzător vârfului `x`, dacă `x≠start`;
- 2, dacă `x=start`.

Se observă că toate arcele au costul strict pozitiv.

Problema se reduce la determinarea lungimii drumului minim de la `start` la `stop`, a punctelor care se află pe cel puţin un drum minim şi a numărului de drumuri minime.

Lungimea drumurilor minime de la `start` la toate vârfurile o determinăm folosind algoritmul Dijkstra, modificat astfel încât să păstreze toţi taţii pentru fiecare vârf (nu doar unul).

Fie acum un graf orientat G_2, care are aceleaşi vârfuri ca şi G_1 şi în care arcele sunt legăturile `(fiu,tata)` determinate de algoritmul lui Dijkstra. Graful G_2 este aciclic.

Determinarea vârfurilor care sunt pe cel puţin un drum minim de la `start` la `stop` în G_1 se reduce la determinarea vârfurilor la care se poate ajunge plecând din `stop` în G_2.

Determinăm numărul drumurilor minime de la `start` la `stop` în G_1 care este egal cu numărul drumurilor de la `stop` la `start` în G_2. Pentru fiecare vârf `istart` notăm numărul drumurilor de la `i` la `start` în G_2 cu `drumuri(i)` şi avem:

`drumuri(i)=suma(drumuri(j))`, pentru toţi `j`, taţi ai lui `i`, dacă `start` nu este tată al lui `i`, respectiv

`drumuri(i)=1+suma(drumuri(j))`, pentru toţi `j≠start` cu `j` tată al lui `i`, dacă `start` este tată al lui `i`.

Calculăm `drumuri(stop)` cu ajutorul acestei formule, folosind metoda programării dinamice.

Timpul de executare este dominat de timpul de executare al algoritmului Dijkstra, iar în condiţiile concursului nu a fost nevoie de o implementare Dijkstra mai bună de $O(n^2)$, unde `n` este numărul de vârfuri. Să observăm că grafurile G_1 şi G_2 au o proprietate importantă: *fiecare vârf este adiacent cu cel mult 4 vârfuri, deci numărul de arce e majorat de o constantă înmulţită cu numărul de vârfuri.* Ţinând cont de aceasta, algoritmul Dijkstra poate fi implementat în timpul $O(n*\log n)$.

Listing: DRUMURI.PAS

```
{$b-,e+,n+}
const maxlin=35; maxcol=85;
      maxnod=maxlin*maxcol;
type vecint=array[1..maxnod] of Integer;
     vec4int=array[1..maxnod,1..4]of Integer;
     veclong=array[1..maxnod] of Longint;
     vecbool=array[1..maxnod] of Boolean;
     vecdouble=array[1..maxnod] of Double;
var a:array[1..maxlin] of string[maxcol];
    nvecini:vecint;
    vecini,cost:vec4int;
    nlin,ncol,n,start,stop:Integer;

function punct2vf(lin,col:Integer):Integer;
begin
  punct2vf:=ncol*(lin-1)+col
end;
procedure vf2punct(i:Integer;
                   var lin,col:Integer);
begin
  lin:=((i-1) div ncol)+1;
  col:=((i-1) mod ncol)+1
end;

procedure citire;
var i,j,k,l,cc:Integer;
    gasit:Boolean;
begin
  nlin:=0;
  repeat
    Inc(nlin);
    Readln(a[nlin])
  until a[nlin,1]='_';
  Dec(nlin);
  ncol:=Length(a[1]);
  n:=nlin*ncol;
  gasit:=false;
  for i:=1 to nlin do
    for j:=1 to ncol do
      if a[i,j]=#1
      then if not gasit
           then begin
                  start:=punct2vf(i,j);
                  gasit:=true
                end
           else stop:=punct2vf(i,j);
  Fillchar(nvecini,Sizeof(nvecini),0);
  Fillchar(vecini,Sizeof(vecini),0);
  Fillchar(cost,Sizeof(cost),0);
  for i:=1 to nlin do
    for j:=1 to ncol do
      if a[i,j]<>#219
      then
      begin
        k:=punct2vf(i,j);
```

```
        if (i>1) and (a[i-1,j]<>#219)
        then
        begin
          Inc(nvecini[k]);
          vecini[k,nvecini[k]]:=
                              punct2vf(i-1,j)
        end;
        if (i<nlin) and (a[i+1,j]<>#219)
        then
        begin
          Inc(nvecini[k]);
          vecini[k,nvecini[k]]:=
                              punct2vf(i+1,j)
        end;
        if (j>1) and (a[i,j-1]<>#219)
        then
        begin
          Inc(nvecini[k]);
          vecini[k,nvecini[k]]:=
                              punct2vf(i,j-1)
        end;
        if (j<ncol) and (a[i,j+1]<>#219)
        then
        begin
          Inc(nvecini[k]);
          vecini[k,nvecini[k]]:=
                              punct2vf(i,j+1)
        end;
        if a[i,j]=#1
        then cc:=1
        else cc:=1+Ord(a[i,j])-175;
        if k=start
        then cc:=2;
        for l:=1 to nvecini[k] do
          cost[k,l]:=cc
      end
  end;

var ales:^vecbool;
    ntati:^vecint;
    tati:^vec4int;
    minim:^veclong;

procedure alege(i:Integer);
var j,k:Integer;
begin
  ales^[i]:=true;
  for j:=1 to nvecini[i] do
  begin
    k:=vecini[i,j];
    if minim^[k]<minim^[i]+cost[i,j]
    then Continue;
    if minim^[k]>minim^[i]+cost[i,j]
    then
    begin
      minim^[k]:=minim^[i]+cost[i,j];
      ntati^[k]:=1; tati^[k,1]:=i;
      Continue
    end;
    Inc(ntati^[k]);
    tati^[k,ntati^[k]]:=i
  end;
end;

var ndrumuri:^vecdouble;
function drumuri(i:Integer):Double;
var j,k:Integer;
begin
  if ndrumuri^[i]<0
  then
    if i=start
    then ndrumuri^[i]:=1
    else
    begin
      vf2punct(i,j,k);
      a[j,k]:=' ';
      ndrumuri^[i]:=0;
      for j:=1 to ntati^[i] do
      ndrumuri^[i]:=ndrumuri^[i]+
                          drumuri(tati^[i,j])
    end;
  drumuri:=ndrumuri^[i]
end;

var i,j,k:Integer;
Begin
  Assign(input,'D.DAT'); Reset(input);
  New(minim); New(ntati); New(tati);
  New(ales); New(ndrumuri);
  while not Seekeof do
  begin
    citire;
    Fllchar(ales^,Sizeof(ales^),0);
    Fillchar(ntati^,Sizeof(ntati^),0);
    for i:=1 to n do
      minim^[i]:=MaxLongint div 2;
    minim^[start]:=0;
    alege(start);
    for i:=1 to n-1 do
    begin
      j:=0;
      for k:=1 to n do
        if not ales^[k]
        then
          if j=0
          then j:=k
          else if minim^[j]>minim^[k]
                 then j:=k;
      if minim^[j]=MaxLongint div 2
      then Break;
      alege(j)
    end;
```

```
    if minim^[stop]=MaxLongint div 2
    then Writeln('0 drumuri, cost 0')
    else
    begin
      for i:=1 to n do ndrumuri^[i]:=-1;
      Writeln(drumuri(stop):0:0,
              ' drumuri, cost ',minim^[stop]);
      vf2punct(stop,j,k); a[j,k]:=#1
    end;
    for i:=1 to nlin+1 do Writeln(a[i])
  end;
  Close(input)
End.
```

P029905: Inversiuni

Deși există algoritmi (destul de complicați) de complexitate $O(n* log\ n)$ pentru această problemă, ținând cont de restricția $n \leq 30000$ și de timpul de rulare cerut la concurs, este suficient algoritmul banal de ordinul $O(n^2)$, care necesită calcule puțin optimizate.

Listing: INVERS.PAS

```
{$r-}
var n,i,j,ai:Integer;
    ninv:Longint;
    a:array[1..32000] of Integer;
Begin
  Assign(input,'E.DAT'); Reset(input);
  while not SeekEof do
  begin
    Read(n); ninv:=0;
    for i:=1 to n do Read(a[i]);
    for i:=1 to n-1 do
    begin
      ai:=a[i];
      for j:=i+1 to n do
        if ai>a[j] then Inc(ninv)
    end;
    Writeln(ninv)
  end;
  Close(input)
End.
```

P029906: Rețea de comunicare

Cea mai complicată parte din această problemă este citirea și memorarea intrării (jumătate din enunț se referă doar la formatul de introducere a datelor). Pentru fiecare comutator am folosit o structură SG care conține:

`l` - comutatorul legat la intrarea `L`;
`lc`- ieșirea comutatorului memorat în `l`, legată de intrarea `L`;
`d` - comutatorul legat la intrarea `D`;
`dc`- ieșirea comutatorului memorat în `d`, legată de intrarea `D`;
`s` - starea pinului de control al comutatorului.

De o mare importanță este observația că *nu pot exista cicluri* (din enunț `COM2>COM1` pentru fiecare conectare). Din acest motiv se pot calcula direct valorile dorite pentru diferite ieșiri ale comutatoarelor. Aceasta se face prin identificarea repetată a tatălui până când se ajunge la o intrare neconectată, iar apoi se verifică semnalul pentru aceasta.

Listing: RETEA.PAS

```
{$M 16384,0,655360}
type SG = record
            l,d:Integer;
            lc,dc:Char; s:Byte
          end;
const max = 4010;
var a:array[0..max] of SG;
    ntest,n,testfor:Integer;
    b:array[0..max,1..2] of Byte;

procedure tata(sw:Integer; loc:Char;
          var ttata:Integer; var leg:Char);
begin
  if a[sw].S=0
  then if loc='R' then leg:='L' else leg:='D'
  else if loc='R' then leg:='D'
                  else leg:='L';
  if leg='L' then ttata:=a[sw].L
             else ttata:=a[sw].D;
  if ttata=-1 then Exit;
  if leg='L' then leg:=a[sw].lc
             else leg:=a[sw].dc
end;

procedure doit(sw:Integer; loc:Char);
var t:Integer; w:Char;
begin
  repeat
    tata(sw,loc,t,w);
    if t<>-1 then begin loc:=w; sw:=t end;
  until t=-1;
  if a[sw].S=0
  then if loc='R' then loc:='L'
                  else loc:='D'
  else if loc='R' then loc:='D'
                  else loc:='L';
  if loc='L' then Write(b[sw,1],' ')
             else Write(b[sw,2],' ')
end;

procedure init;
var i,nconex,sw1,sw2,ns1,nteste:Integer;
    c1,c2:Char;
begin
  for i:=0 to max do
  begin
    a[i].L:=-1; a[i].D:=-1; a[i].s:=0
  end;
  Fillchar(b,Sizeof(b),0);
  Readln(n,nconex);
```

```
  for i:=1 to nconex do
  begin
    Read(sw1);
    repeat Read(c1) until c1<>' ';
    Read(sw2);
    repeat Read(c2) until c2<>' ';
    if c2='L'
    then
    begin
      a[sw2].L:=sw1;
      a[sw2].Lc:=c1
    end
    else
    begin
      a[sw2].D:=sw1;
      a[sw2].Dc:=c1
    end;
    Readln;
  end;
  for i:=0 to n-1 do Read(a[i].S);
  Readln;
  Readln(ns1);
  for i:=1 to ns1 do
  begin
    Read(sw1);
    repeat Read(c1) until c1<>' ';
    if c1='L'
    then b[sw1,1]:=1
    else b[sw1,2]:=1;
    Readln
  end;
  Readln(nteste);
  for i:=1 to nteste do
  begin
    Read(sw1);
    repeat Read(c1) until c1<>' ';
    doit(sw1,c1);
    Readln
  end;
  Writeln; Readln
end;

Begin
  Assign(input,'F.DAT'); Reset(input);
  Readln(ntest);
  for testfor:=1 to ntest do init;
  Close(input)
End.
```

P029907: Combinații

Prima etapă a rezolvării constă în adaptarea șirului citit la lungimea cerută, adică înlocuirea sa cu un șir de această lungime, astfel încât rezultatele să fie aceleași. Dacă șirul e prea scurt, acesta se completează cu cea mai mică literă considerată (adică 'A'), ceea ce e valabil și pentru inferior și pentru superior. Dacă șirul e prea lung, atunci pentru inferior este suficientă trunchierea șirului, deoarece șirul trunchiat e mai mic (lexicografic) decât șirul inițial; însă pentru superior șirul trebuie incrementat în urma trunchierii. Incrementarea are loc exact ca la numerele naturale:

aZ+1=aa; xzCzzz+1=xzDAAA; zzzzz+1=(null).

Urmează prelucrarea șirului obținut. Atât șirul inferior, cât și șirul superior trebuie să se încadreze literă cu literă între cel mai mic șir și cel mai mare șir (în ordine lexicografică) cu k litere. De exemplu, pentru k=4 aceste șiruri sunt ABCD și wxyz. De aceea vom face 3 teste:

- "testul sus" verifică depășirea celui mai mare șir;
- "testul jos" verifică depășirea ("în jos") a celui mai mic șir;
- "testul de ordine" verifică ordonarea strict crescătoare a literelor.

Pentru determinarea inferiorului:

"Testul jos" se termină în sens negativ (nu are loc depășire) dacă ajungem la o literă strict mai mare decât litera corespunzătoare din șirul cel mai mic, deoarece în continuare va fi aplicat "testul de ordine".

Dacă ajungem la o literă pentru care "testul sus" dă rezultat pozitiv (depășire), atunci copiem cel mai mare șir de la poziția respectivă până la sfârșitul șirului. "Testul de ordine" pentru poziția curentă este automat verificat, deoarece litera precedentă a verificat "testul sus".

În sfârșit, dacă litera curentă nu verifică "testul de ordine", atunci decrementăm subșirul format din literele anterioare (păstrând ordinea literelor!) și apoi copiem cel mai mare șir din poziția curentă până la sfârșit. Decrementarea este posibilă dacă a fost trecut "testul jos".

Pentru determinarea superiorului, lucrurile sunt ceva mai simple:

"Testul jos" este automat verificat de "testul de ordine".

"Testul sus" este necesar doar dacă prima literă a șirului este mai mare sau egală cu prima litera a celui mai mare șir; în caz contrar este asigurată posibilitatea incrementării șirului, păstrând ordinea literelor și "testul sus" va fi suplinit de "testul de ordine".

Dacă litera curentă nu verifică "testul de ordine", atunci incrementăm subșirul format din literele anterioare cu ultima literă dublată (păstrând ordinea literelor și verificând "testul sus") și apoi completăm șirul cu litere consecutive.

Algoritmul este evident liniar în k, deci practic are timp constant (k≤52).

Listing: COMBIN.PAS

```
var i,k:Word;
    x:string;
    litere:array[1..52]of Char;

function lit_urm(x:Char):Char;
begin
  if x<>'Z'
  then lit_urm:=Succ(x)
  else lit_urm:='a'
end;
```

```
function lit_prec(x:Char):Char;
begin
  if x<>'a'
  then lit_prec:=Pred(x)
  else lit_prec:='Z'
end;

function inferior(k:Word; x:string):string;
var i,j:Word;
    testjos:Boolean;
begin
  inferior:='null';
  if Length(x)>k
  then x:=Copy(x,1,k)              { trunchiere }
  else
    for i:=1 to k-Length(x) do x:=x+'A';
                                   { completare }
  testjos:=true;
  for i:=1 to k do
  begin
    if testjos
    then begin                     { test jos: }
           if x[i]<litere[i] then Exit;
           if x[i]>litere[i]
           then testjos:=false;
         end;
    if x[i]>litere[52+i-k]         { test sus: }
    then begin
           for j:=i to k do
             x[j]:=litere[52+j-k];
           inferior:=x;
           Exit
         end;
    if i>1
    then                           { test de ordine: }
      if x[i]<=x[i-1]
      then
      begin
        repeat Dec(i)
        until(i=1)or(x[i]>lit_urm(x[i-1]));
        x[i]:=lit_prec(x[i]);
                                   { decrementare }
        for j:=i+1 to k do
          x[j]:=litere[52+j-k];
        inferior:=x;
        Exit
      end
  end;
  inferior:=x
end;

function superior(k:Word; x:string):string;
var i,j:Word;
    ok,testsus:Boolean;
begin
  superior:='null';
  if Length(x)>k
  then
  begin
    x:=Copy(x,1,k);                { trunchiere }
    ok:=false;
    for i:=k downto 1 do
      if x[i]<'z'
      then begin ok:=true; Break end;
        if not ok then Exit;
                   { nu putem incrementa }
        x[i]:=lit_urm(x[i]);
        for j:=i+1to k do x[j]:='A';
                                   { incrementare }
      end
  else for i:=1 to k-Length(x) do x:=x+'A';
                                   { completare }
  testsus:=x[1]>=litere[53-k];
  for i:=1 to k do
  begin
    if testsus
    then if x[i]>litere[52+i-k] then Exit;
    if i>1
    then                           { test de ordine: }
      if x[i]<=x[i-1]
      then
      begin
        x[i]:=x[i-1];
        while x[i]>=litere[52+i-k] do
        Dec(i);x[i]:=lit_urm(x[i]);
                                   { incrementare }
        for j:=i+1 to k do
          x[j]:=lit_urm(x[j-1]);
        superior:=x; Exit
      end;
  end;
  superior:=x;
end;

Begin
  for i:=1 to 26 do
  begin
    litere[i]:=Chr(i-1+Ord('A'));
    litere[i+26]:=Chr(i-1+Ord('a'))
  end;
  Assign(input,'G.DAT'); Reset(input);
  while not SeekEof do
  begin
    Read(k);
    if SeekEof then;
    Readln(x);
    Writeln('inferior = ',inferior(k,x));
    Writeln('superior = ',superior(k,x));
    Writeln
  end;
  Close(input)
End.
```

DirectX

Mihai Oltean

Chiar dacă nu sunteți un programator de jocuri, ci doar un simplu jucător, trebuie să fi auzit de DirectX. Această tehnologie este cel mai des folosită în crearea jocurilor pentru sistemul de operare Win95.

Ce este și cum a apărut?

Atuurile oferite de noua tehnologie sunt faptul că oferă o grafică de înaltă viteză și o interfață independentă de hardware. Prima facilitate este clară, dar cea de-a doua necesită câteva explicații. Fiecare producător de plăci video, plăci de sunet, plăci de rețea etc. furnizează propriile drivere care permit manipularea acestor plăci. Un programator, pentru a folosi facilitățile fiecărui produs în parte, va trebui să își adapteze codul scris în funcție de specificațiile furnizate de către fabricanții de hardware (plăci video, plăci de sunet, plăci de rețea). Aici intervine *DirectX* oferind, cum am mai spus, o interfață independentă de hardware. Cu alte cuvinte, programatorul de jocuri nu mai trebuie să țină cont de alcătuirea la nivel de hard a calculatorului, acest lucru fiind implementat în funcțiile API ale *DirectX*-ului de către programatorii de la Microsoft. Pe lângă acestea, proiectarea *DirectX*-ului s-a făcut cu gândul la acceleratoarele hardware (mă refer în special la acceleratoarele grafice). Pentru cei care nu știu sau nu înțeleg cât de important este un astfel de accelerator, am să dau câteva lămuriri. Afișarea unui obiect 3D pe un ecran plat necesită un număr foarte mare de calcule. Efectuarea acestora de către procesor va duce la încetinirea vitezei jocului. Acceleratoarele grafice sunt concepute pentru a prelua o parte din acest tip de muncă efectuată de către procesor. Dacă nu aveți așa ceva, se va face o simulare software.

Înainte de apariția acestei tehnologii, programarea jocurilor se făcea sub MS-DOS folosind spre exemplu compilatorul Watcom C++. Cred că mulți dintre voi vă amintiți că trebuia să puneți o parte din liniile **AUTOEXEC.BAT**-ului în comentariu, deoarece majoritatea jocurilor se foloseau doar de cei 640 Kb de bază. În plus, trebuia scris cod pentru fiecare placă de sunet (sau video) în parte (acest lucru era evident, deoarece erați solicitați să alegeți, dintr-un meniu, configurația propriului calculator). Acum, odată cu lansarea tehnologiei **DirectX** (de către Microsoft) pentru Win95 toate aceste probleme au dispărut. Dar, înainte de a continua, vom defini **DirectX**. Precum ați remarcat din rândurile anterioare, nu este vorba nici de un limbaj, nici de un mediu de programare, nici de un echipament hard, ci pur și simplu de un set de funcții API. Asemănarea dintre *X*-ul lui *DirectX* și *X*-ul lui *ActiveX* nu este pur întâmplătoare. Ambele se bazează pe obiecte *COM*, despre care voi vorbi puțin mai târziu.

Toate funcțiile necesare acestei tehnologii sunt stocate în câteva *dll*-uri care se găsesc în directorul `\Windows\System`. Dintre acestea aș enumera **`ddraw.dll`**, **`dsound.dll`**, **`dplay.dll`**, **`dinput.dll`** etc. Pe măsură ce *DirectX*-ul este extins, numărul *dll*-urilor va fi probabil în continuă creștere. În prezent s-a ajuns la versiunea 6.2.

Implementarea unui joc nu mai este limitată acum la un singur limbaj de programare (C++). Medii precum Visual C++, C++Builder, Delphi și chiar Visual Basic au implementate headere ale funcțiilor *DirectX*. Oricum, *DirectX*-ul original a fost implementat pentru medii bazate pe C și C++, Microsoft-ul oferind doar pentru aceste headere și documentații. În continuare ne vom folosi de C++, mai precis de mediul C++Builder al firmei Borland (în prezent *Inprise Corporation*). Utilizatorii lui Visual C++ nu trebuie să facă decât un minim de transformări.

În legătură cu cantitatea de informație din acest articol, consider că este suficientă pentru a crea un joc simplu (nici Starcraft-ul nu ar fi greu de realizat dacă ați avea la dispoziție imaginile și motorul de joc). Voi prezenta doar noțiunile legate de grafică, deoarece sunt cele mai importante. Înțelegerea lor simplifică înțelegerea celorlalte componente.

DirectX și COM

Am amintit că există o legătură între obiectele care alcătuiesc *DirectX*-ul și obiectele *COM*. Pentru cei care nu știu ce înseamnă *COM* și la ce este util, am să dau câteva lămuriri. *COM* (*Component Object Model*) este o tehnologie care permite obiectelor să interacționeze dincolo de granițele aplicației. Cu alte cuvinte este posibil ca un obiect să interacționeze cu un alt obiect, chiar dacă acesta se află în interiorul unei alte aplicații. Un obiect *COM* este o cutie

neagră. Manipularea datelor din interiorul lui se face cu ajutorul metodelor. Este posibil ca un obiect *COM* să aibă mai multe metode, de obicei grupate. Un astfel de grup (o submulțime de metode) poartă numele de ***interfață***. Deci un obiect *COM* poate avea una sau mai multe interfețe, fiecare dintre acestea fiind alcătuite din una sau mai multe metode (sau funcții).

Toate obiectele *COM* derivă din **IUnknown**, care se ocupă cu manipularea timpului de viață al unui obiect și cu navigarea între interfețe. Metodele definite de **IUnknown** sunt trei la număr, și anume: **AddRef**, **Release** și **QueryInterface**. Primele două tratează durata de viață a obiectului. Atunci când este creat un astfel de obiect numărul de referințe la el va fi setat la 1. Pe măsură ce alte obiecte se atașează de el (adică obțin un pointer la una din interfețe) este apelată metoda **AddRef** care incrementează cu 1 numărul referințelor. Când terminați de folosit o interfață, trebuie să apelați metoda `Release` care va decrementa cu 1 numărul de referințe la obiect. Când numărul de referințe la obiect va fi 0, atunci el va fi eliberat din memorie. Metoda **QueryInterface** testează un obiect *COM* dacă are sau nu o anumită interfață. Dacă da, atunci este întors un pointer la ea.

Să studiem utilitatea acestor metode în cazul obiectelor DirectX. În primul rând metodele **AddRef** și **QueryInterface** nu trebuie apelate într-un simplu joc, deoarece funcțiile din *DirectX* au singure în grijă incrementarea numărului de referințe și navigarea printre interfețe. Singura metodă care trebuie apelată este `Release`. În caz contrar veți avea goluri în memorie.

DirectDraw

Acum sunteți pregătiți pentru a vă explica în ce constă esența programării cu *DirectDraw*. Vom numi ceea ce vedeți pe ecran ***suprafață primară***. Pe ea se află desenate tot felul de obiecte. Tehnologia *DirectDraw* ne permite ca la un moment dat să avem mai multe suprafețe. Doar una dintre ele va fi suprafață primară, deoarece doar una poate fi afișată la un moment dat pe ecran. Celelalte vor fi numite ***suprafețe din spate*** și ele nu sunt afișate pe ecran. Dar la un moment dat se poate face o interschimbare de suprafețe (între suprafața primară și o suprafață din spate) astfel că suprafața care era afișată va ajunge suprafață din spate, iar suprafața care a fost în spate va ajunge suprafață primară. Desenarea pe o suprafață se face doar când aceasta este în spate. După ce această desenare este efectuată, se poate face interschimbarea între suprafețe, deci astfel se poate realiza animația. Pentru a înțelege cum se face această rotire de suprafețe vom încerca să definim ***suprafața***. O suprafață este o zonă de memorie. Când se face o schimbare de suprafețe, are loc o schimbare de zone de memorie. Mai precis are loc doar o schimbare a locului spre care indică cei doi pointeri la suprafețe. Dacă ambele suprafețe sunt stocate în memoria video, atunci schimbarea este instantanee. Dar pentru ca acest lucru să fie posibil este nevoie de o mare cantitate de memorie video. De exemplu pentru a stoca două suprafețe de 640x480 cu 256 de culori, va fi nevoie de 614.400 de octeți, asta fără să mai socotim și eventualele bitmap-uri de care are nevoie jocul nostru. În cazul în care această memorie nu este suficientă, stocarea unei suprafețe se va face în memoria RAM, ceea ce va duce la o oarecare încetinire a procesului de interschimbare de suprafețe.

Obiectele din care este alcătuit *DirectDraw* sunt:

IDirectDraw,
IDirectDrawSurface,
IDirectDrawPalette și
IDirectDrawClipper.

Vom trata, pe rând, fiecare dintre aceste obiecte.

IDirectDraw este principalul obiect al unei aplicații *DirectX*. El reprezintă dispozitivul de afișare (adică monitorul, dacă facem o aproximație grosolană). Într-o aplicație normală este nevoie doar de un singur obiect `IDirectDraw`, dar în prezent calculatoarele cu mai multe monitoare sunt din ce în ce mai frecvente.

IDirectDrawSurface reprezintă o suprafață primară sau o suprafață din spate. Principala metodă a acestui obiect este `Flip`, care efectuează schimbarea între suprafețe.

IDirectDrawPalette reprezintă o paletă de culori. O ***paletă de culori*** este un vector care conține valorile culorilor ce pot fi afișate curent de ecran. De exemplu, în modul cu 256 de culori, valoarea unui pixel de ecran este de fapt un index în tabelul (paleta) în care sunt stocate

Componentele DirectX

Nume	Descriere
DirectDraw	Furnizează acces la echipamentul de afișare (plăci video). Manipulează suprafețe, rotații de pagini etc.
DirectSound	Furnizează acces la echipamentul de sunet (plăci de sunet).
DirectPlay	Furnizează un protocol independent pentru comunicarea între calculatoare. Se poate comunica prin Internet, rețele locale... până la cabluri seriale.
DirectInput	Manipulează intrarea de la diferite echipamente precum mouse, tastatură, joystick.
Direct3D	Furnizează funcții API pentru grafică 3D.

valorile culorilor.

IDirectDrawClipper este util în momentul în care se dorește rularea aplicației într-o fereastră și nu pe tot ecranul. În acest caz există posibilitatea ca ferestrele să se suprapună (nu neapărat integral), funcțiile *DirectX* putând desena și în afara spațiului destinat lor. Prin atașarea unei ferestre unui obiect Clipper și apoi atașarea acestui obiect la o suprafață, toate aceste probleme vor dispărea.

Pașii necesari și variabilele folosite într-o aplicație DirectX

- Inițializarea obiectului **IDirectDraw**
- Setarea aplicației pentru a rula pe tot ecranul sau numai într-o fereastră
- Setarea rezoluției monitorului
- Crearea unei suprafețe primare și a mai multor suprafețe din spate
- Desenarea pe suprafețele din spate
- Schimbarea între suprafețele din spate și suprafața primară pentru a se obține animația

Variabilele folosite într-o aplicație *DirectX* sunt următoarele:

```
LPDIRECTDRAW lpDD;  // obiectul DirectDraw
LPDIRECTDRAWSURFACE lpDDSPrimary;
                    // suprafața primară
LPDIRECTDRAWSURFACE lpDDSBack;
                    // suprafața din spate
BOOL bActive;       // este aplicația activă
BYTE faza;
```

Pentru a rula această aplicație mai avem nevoie de fișierul cu prototipurile funcțiilor și obiectelor *DirectDraw*. Acesta se numește **ddraw.h** și este inclus în toate versiunile mediului C++Builder.

Inițializarea DirectDraw-ului

Inițializarea obiectului **IDirectDraw** se face cu ajutorul funcției **DirectDrawCreate** care are trei parametri: primul reprezintă dispozitivul de afișare și trebuie setat la **NULL** (folosindu-se astfel dispozitivul implicit), al doilea este adresa pointerului spre obiectul **IDirectDraw** ce urmează a fi creat și al treilea parametru este nefolosit și trebuie setat la **NULL**. Dacă totul a decurs bine, funcția întoarce valoarea DD_OK, iar în caz contrar o eroare. Codul sursă care realizează inițializarea este următorul:

```
ddrval=DirectDrawCreate(NULL,&lpDD,NULL);
if(ddrval == DD_OK)
{ //lpDD este un obiect valid }
else
{ //obiectul DirectDraw nu a putut fi creat }
```

După inițializare trebuie specificat în cel fel de fereastră rulează aplicația. Acest lucru se face prin intermediul funcției **SetCooperativeLevel**, care are doi parametri. Primul reprezintă fereastra aplicației și al doilea indică modul în care va rula aplicația. Se poate alege între rulare pe tot ecranul (DDSCL_FULLSCREEN) sau rulare într-o fereastră (DDSCL_NORMAL). Dacă se alege rulare pe tot ecranul atunci se poate obține acces exclusiv (DDSCL_EXCLUSIVE) asupra resurselor, în sensul că nici o altă aplicație nu mai are voie să redeseneze ecranul. Codul sursă care setează aplicația pentru rulare în mod exclusiv pe tot ecranul, este următorul:

```
ddrval = lpDD->SetCooperativeLevel(hwnd,
          DDSCL_EXCLUSIVE |DDSCL_FULLSCREEN);
if(ddrval==DD_OK)
{ // ok }
else
{ // insucces, s-ar putea totuși
  // ca aplicație să mai ruleze
}
```

Acum trebuie să indicăm rezoluția și numărul culorilor disponibile. Ne folosim de funcția **SetDisplayMode** care are trei parametri: primii doi indică numărul de pixeli pe orizontala, respectiv verticala ecranului, iar al treilea indică numărul de biți necesari pentru a stoca culoarea unui pixel. Următorul cod setează rezoluția la 640×480 cu 256 culori:

```
ddrval=lpDD->SetDisplayMode(640,480,8);
if(ddrval==DD_OK)
{ // s-a schimbat modul }
else
{
// modul nu poate fi schimbat
// deoarece fie nu este suportat de către placa
// video, fie altcineva a obținut mod exclusiv
}
```

Crearea suprafețelor

În primul rând trebuie completate câmpurile unei structuri DDSURFACEDESC. Cele mai importante câmpuri sunt dwSize (care este folosit pentru a permite viitoare extinderi ale acestei structuri), dwFlags (care indică membrii structurii care se vor inițializa), ddsCaps (care va indica tipul suprafeței create) și dwBackBufferCount (care contorizează numărul suprafețelor din spate, atașate suprafeței primare). Dacă dorim o suprafață primară care are atașată o singură suprafață din spate, atunci modul în care este completată structura DDSURFACEDESC este următorul:

```
DDSURFACEDESC ddsd;
ddsd.dwSize=sizeof(ddsd);
ddsd.dwFlags=DDSD_CAPS|DDSD_BACKBUFFERCOUNT;
ddsd.ddsCaps.dwCaps=DDSCAPS_PRIMARYSURFACE|
                    DDSCAPS_FLIP|DDSCAPS_COMPLEX;
ddsd.dwBackBufferCount=1;
```

În continuare, trebuie să apelăm metoda `CreateSurface` pentru a crea suprafețele dorite. Această metodă are ca parametri o structură de tip `DDSURFACEDESC`, un pointer la suprafața primară și un al treilea parametru care trebuie setat la **`NULL`**:

```
ddrval=lpDD->CreateSurface(&ddsd,
                           &lpDDSPrimary,NULL);
if(ddrval==DD_OK)
{
//lpDDSPrimary pointează spre suprafața primară
}
else
{ // suprafața nu a fost creată
  return FALSE;
}
```

Am amintit anterior că suprafața primară are atașată o suprafață din spate. Pentru a obține un pointer la ea, ne folosim de funcția `GetAttachedSurface`:

```
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
ddrval=lpDDSPrimary->GetAttachedSurface
                        (&ddcaps,&lpDDSBack);
if(ddrval==DD_OK)
{
//lpDDSBack pointează spre suprafața din spate
}
else { return FALSE; }
```

Desenarea pe suprafețe

Desenarea se realizează folosind funcțiile Windows standard. Mai întâi trebuie obținut un *handler* la un context de dispozitiv, iar apoi cu ajutorul acestuia se pot efectua modificări asupra suprafețelor. După terminarea desenării trebuie apelată metoda `ReleaseDC` care va debloca accesul la suprafața respectivă. Codul sursă care efectuează desenarea pe suprafețe a două șiruri de caractere este următorul:

```
if(lpDDSPrimary->GetDC(&hdc)==DD_OK)
{ SetBkColor(hdc,RGB(0,0,255));
  SetTextColor(hdc,RGB(255,255,0));
  TextOut(hdc,100,100,'textul din fata',16);
  lpDDSPrimary->ReleaseDC(hdc);
}

if(lpDDSBack->GetDC(&hdc)==DD_OK)
{ SetBkColor(hdc,RGB(0,0,255));
  SetTextColor(hdc,RGB(255,255,0));
  TextOut(hdc,0,0,'textul din spate',16);
  lpDDSBack->ReleaseDC(hdc);
}
```

Rotirea suprafețelor

Se realizează cu ajutorul metodei **`Flip`**. Pot apărea câteva situații speciale care trebuie tratate, și anume:

- Este posibil ca memoria video, datorită faptului că mai există și alte aplicații în sistem, să fie eliberată. În aceste condiții trebuie apelată metoda **`Restore`** pentru a recupera suprafața respectivă.
- O altă problemă care apare este cauzată de *refresh*-ul monitoarelor, prin acesta înțelegând de câte ori pe secundă se redesenează ecranul. Timpul scurs între două redesenări se numește ***spațiu vertical***. Este posibil ca interschimbarea anterioară a suprafețelor să nu fi avut loc din cauza acestor spații verticale. În acest caz este întoarsă valoarea `DDERR_WASSTILLDRAWING`, care înseamnă că metoda **`Flip`** va trebui apelată în continuare până când se returnează `DD_OK`.

```
while( 1 )
{
  HRESULT ddrval;
  ddrval=lpDDSPrimary->Flip(NULL,0);
  if(ddrval==DD_OK) { break; }
  if( ddrval==DDERR_SURFACELOST)
  { ddrval=lpDDSPrimary->Restore();
    if(ddrval!=DD_OK)
    { break;
    }
  }
 if(ddrval!=DDERR_WASSTILLDRAWING) { break; }
}
```

Referitor la această rotire de suprafețe, mai trebuie spus faptul că ea se face de obicei în interiorul unui *Timer*. Începătorii pot folosi componenta **`TTimer`** din C++Builder, dar pentru o mai bună acuratețe trebuie folosite timerele multimedia (definite în unit-ul **`MMSYSTEM.H`**). Înainte de efectuarea rotirii (descrisă cu câteva rânduri mai sus), mai trebuie desenate textele respective pe suprafața din spate, în funcție de faza în care se află aplicația. Astfel, dacă faza este 0, atunci se desenează *textul din spate*, iar în caz contrar se desenează *textul din față*. Abia după această desenare se poate face rotirea suprafețelor.

Eliberarea obiectelor DirectDraw

Eliberarea se face după indicațiile din partea despre obiectele *COM*:

```
if(lpDD!=NULL) {
  if(lpDDSPrimary!=NULL)
  { lpDDSPrimary->Release();
    lpDDSPrimary=NULL;
  }
  lpDD->Release(); lpDD=NULL;
}
```

Mihai Oltean este student la Facultatea de Matematică, secția Informatică, Universitatea Babeș-Bolyai, Cluj. Poate fi contactat prin e-mail la adresa: om7280@cs.ubbcluj.ro

BACALAUREAT 1999

Doina Rancea

Se apropie BAC-ul... Nu lăsați pregătirea examenului de info pe ultima clipă! Aveți nevoie doar de recapitularea unor cunoștințe pe care le-ați acumulat când erați un pic mai "tineri".

Subiectul I

Programul afișează descompunerea unui număr natural n≤65535 în factori primi.

Subiectul II

Programul afișează cifrele unui număr natural în ordine inversă, pornind de la cifra unităților.

Listing: CIFRE.PAS

```
Program Afisare_cifre_nr_in_ordine_inversa;
type natural=0..MaxLongint;
var n:natural;
    c:Byte;
Begin
  Write('n=');
  Readln(n);
  repeat
    c:=n mod 10;
    Write(c);
    n:=n div 10;
  until n=0;
  Readln
End.
```

Subiectul III

Metoda *Divide et Impera* constă în descompunerea problemei date în subprobleme independente care operează asupra unui set de date având mărimea din ce în ce mai mică. De asemenea, la fiecare revenire din rezolvarea unei subprobleme trebuie asigurată combinarea soluțiilor subproblemelor urmând ca, în final, să se obțină soluția problemei inițiale.

Prezentăm o implementare recursivă a algoritmului care rezolvă problema dată.

Listing: MINIM.PAS

```
Program Minim_prin_Divide_et_impera;
var x:array[1..50] of Integer;
    i,n:Byte;
Function Minim(st,dr:Byte):Integer;
var min1,min2:Integer;
    mijloc:Byte;
begin
  case dr-st of { rezolvarea subproblemelor }
    0: Minim:=x[st];   { de dimensiune mica }
    1: if x[st]<x[dr] then Minim:=x[st]
                      else Minim:=x[dr];
  else
  begin
  { descompunerea problemei Minim(st,...,dr) }
  { in subproblemele: }
                  { Minim(st,...,mijloc) }
                  { Minim(mijloc+1,...,dr) }
    mijloc:=(st+dr) div 2;
    min1:=Minim(st,mijloc);
    min2:=Minim(mijloc+1,dr);
    if min1<min2
    then Minim:=min1
    { combinarea solutiilor subproblemelor }
    else Minim:=min2
  end
  end
end;

Begin
  Write('n='); Readln(n);
  Write('numerele:');
  for i:=1 to n do Read(x[i]);
  Readln;
  Writeln('Minim=',Minim(1,n));
              { apel Minim cu: st=1, dr=n }
  Readln
End.
```

Subiectul IV

Enunțul problemei precizează în detaliu toate cerințele, drept urmare, soluția care urmează nu necesită explicații suplimentare.

Listing: DICTION.PAS

```
Program Dictionar;
type reper=^elem;
     elem=record
            cuv_rom,cuv_engl:string[30];
            urm:reper;
          end;
var sant,ult:reper;

Function Nu_exista(cuv:string):Boolean;
var p:reper;
begin
  p:=sant^.urm;
  while p<>nil do
    if cuv=p^.cuv_rom
    then begin Nu_exista:=false; Exit end
    else p:=p^.urm;
  Nu_exista:=true
end;

Procedure Creare;
var p:reper;
    cuv1,cuv2:string[30];
begin
  New(sant); sant^.urm:=nil; ult:=sant;
  Write('Cuv. in romana: ');
  while not Eof do
  begin
    Readln(cuv1);
    Write('cuv in engleza: '); Readln(cuv2);
    if Nu_exista(cuv1)
    then
    begin
      New(p);   { alocare mem. pt. un elem. }
      p^.cuv_rom:=cuv1;      { initializare }
      p^.cuv_engl:=cuv2;
      p^.urm:=nil;
      ult^.urm:=p;   { legare in structura }
      ult:=p
    end;
    Write('Cuv. în romana: ')
  end
end;

Procedure Listare;
var p:reper;
begin
  p:=sant^.urm;
  Writeln('Cuvintele din dictionar sunt:');
  While p<>nil do
  begin
    Writeln(p^.cuv_rom,'-',p^.cuv_engl);
    p:=p^.urm
  end
end;

Begin
  Creare;
  Listare
End.
```

Subiectul V

Fișierul de intrare cuprinde datele unei singure clase cu cel mult 50 de elevi; există astfel posibilitatea păstrării într-un tablou a tuturor datelor elevilor promovați. Pentru afișarea listei cerute, tabloul s-a ordonat prin *metoda bulelor*. Corectitudinea datelor din fișier, asigurată de enunțul problemei, a simplificat mult rezolvarea.

Listing: PROMO.PAS

```
Program Promovati;
type art=record
           nume,pren:string[30];
           med:Byte;
         end;
var f:Text;
    c:Char;
    elev:array[1..50] of art;
    aux:art;
    ordonat:Boolean;
    n,i:Byte;

Begin
  Assign(f,'C:\MEDII.TXT'); Reset(f);
  n:=0;
  while not Eof(f) do
  begin
    Inc(n);
    with elev[n] do
    begin
      nume:=''; pren:=''; Read(f,c);
      while c<>' ' do
      begin nume:=nume+c; Read(f,c) end;
      Read(f,c);
      while c<>' ' do
      begin pren:=pren+c; Read(f,c) end;
      Readln(f,med)
    end;
    if elev[n].med<5 then Dec(n)
  end;
  Close(f);
  repeat
    ordonat:=true;
    for i:=1 to n-1 do
      if elev[i].med<elev[i+1].med
      then
      begin
        ordonat:=false;
        aux:=elev[i];
        elev[i]:=elev[i+1]; elev[i+1]:=aux
      end;
  until ordonat;
```

```
  Writeln('Elevii promovati:');
  for i:=1 to n do
    with elev[i] do
      Writeln( nume,' ', pren,' ',med);
  Readln
End.
```

Subiectul VI

Problema revine la determinarea unui ciclu hamiltonian minim într-un graf neorientat complet cu n vârfuri. Deoarece nu sunt admise soluții euristice, problema se rezolvă prin *backtracking recursiv*. Graful fiind complet este codificat prin matricea costurilor. Generarea unui traseu este oprită imediat dacă se depășește costul traseului minim curent.

Listing: TRASEU.PAS

```
Program Traseu_minim_de_inspectie;
type natural=0..MaxLongint;
var oras,oras_tr_min: array[1..10] of Byte;
    neinspectat:array[1..10] of Boolean;
    cost:array[1..10,1..10] of natural;
    n,i:Byte;
    c_min,c:natural;

Procedure Citire;
var i,j:Byte;
begin
  Write('Numarul de orase:'); Readln(n);
  for i:=1 to n-1 do
    for j:=i+1 to n do
    begin
      Write('Costul deplasarii intre orasele
                          ',i,' si ',j,' :');
      Readln(cost[i,j]);
      cost[j,i]:=cost[i,j]
    end
end;

Procedure Traseu(i:Byte; c:natural);
var oras_ant,k:Byte;
    cc:natural;
begin
  oras_ant:=oras[i-1];
  for k:=1 to n do
   if neinspectat[k]
  then
  { intreprind. din orasul "k" neinspectata }
  begin
    cc:=c+cost[oras_ant,k];
    if cc>c_min
    then continue;
    oras[i]:=k;
    neinspectat[k]:=false;
    if i<n
    then Traseu(i+1,cc)
    else
    begin        { traseul se inchide in 1 }
      cc:=cc+cost[k,1];
      if cc<c_min
      then
      begin
        c_min:=cc;
        oras_tr_min:=oras
      end
    end;
    neinspectat[k]:=true
  end
end;

Begin
  Citire;
  for i:=1 to n do
    neinspectat[i]:=true;
  oras[1]:=1;
  neinspectat[1]:=false;
  c_min:=MaxLongint;
  Traseu(2,0);
  Writeln('Traseul de inspectie de cost
                      minim=',c_min,' este:');
  for i:=1 to n do
    Write(oras_tr_min[i]:3);
  Writeln(1:3);
End.
```

Punctaj:

Subiectul I:	(0.75 puncte)
Subiectul II:	
Comentarii corecte	(0.25 puncte)
Program corect	(0.50 puncte)
Subiectul III:	
Program corect	(1.25 puncte)
Comentarii corecte	(0.25 puncte)
Subiectul IV:	
Sintaxă corectă	(0.25 puncte)
Procedură corectă de creare a listei	(0.75 puncte)
Funcție corectă de verificare	(0.75 puncte)
Procedură corectă de listare	(0.75 puncte)
Subiectul V:	
Sintaxă corectă	(0.25 puncte)
Citirea corectă a datelor din fișier	(0.50 puncte)
Ordonare descrescătoare după medii	(0.50 puncte)
Listare corectă	(0.25 puncte)
Subiectul VI:	
Citirea costurilor deplasărilor	(0.25 puncte)
Construirea corectă a traseelor	(1.50 puncte)
Determinarea traseului minim	(0.25 puncte)

Doamna profesoară Doina Rancea este cadru didactic la Liceul de Informatică "Tiberiu Popoviciu" din Cluj Napoca; poate fi contactată prin e-mail la adresa: doina@licj.soroscj.ro

PĂRERI și păreri...

Horia Georgescu

Mulți afirmă, nu lipsiți de dezamăgire că Liceul de Informatică nu i-a ajutat cu nimic și că au fost "chinuiți" cu prea multă programare. Pe de altă parte, acest învățământ a pregătit drumul multor programatori excelenți. Dar cum orice drum are un început, să vedem ce ne pot spune studenții din echipa câștigătoare a concursului ACM din toamna anului 1998?

De când studiați informatica? Cum și când a început să vă intereseze programarea?

Cristi Grozea: Am început să "cochetez" cu informatica pe la sfârșitul clasei a VIII-a. Mă interesau mai degrabă calculatoarele decât programarea; la aceasta am ajuns în încercarea de a crea singur programele care îmi lipseau.

Liviu Panait Alexandru: Studiez informatica din școala generală (clasa a VI-a). Am început cu jocurile (ca toată lumea), dar pe CUB-Z, cu monitoare monocolor, apoi am învățat să desenez linii și am perseverat.

Adrian Sandor: Când eram în clasa a VI-a am lucrat pe un minicalculator Independent la un centru de calcul, apoi a venit HC-ul și în final am trecut la PC. Încă de când lucram pe Independent am început să scriu mici programe în Basic; adevărata ucenicie a fost pe HC, iar pe PC am înțeles ce înseamnă programarea adevărată.

Bogdan Stănescu: Eu am "întârziat un pic". Am început doar în clasa a IX-a (1991).

În anii de liceu ați urmat o clasă cu profil de informatică?

Cristi Grozea: Da!

Liviu Panait Alexandru: Am urmat Liceul de Informatică din București deoarece era cel mai "bun" liceu din capitală și trebuie să recunosc că abia aici am început cu adevărat să înțeleg informatica.

Adrian Sandor: Liceul meu avea clase de informatică, dar eu am ales profilul "matematică-fizică", așa că...

Bogdan Stănescu: Da, clasa de informatică de la Liceul "Mihai Viteazul" din Ploiești.

Dacă da, a fost util?

Cristi Grozea: Da.

Bogdan Stănescu: Da, a fost foarte util pentru cei care doreau să învețe informatică.

Ce credeți că ar fi trebuit să se predea în liceu la disciplina informatică?

Cristi Grozea: programare; la Liceul de Informatică din București s-a predat și sper să se mai predea.

Adrian Sandor: Nu pot să răspund; nu am fost elev în clasă de informatică.

Bogdan Stănescu: Conținutul programei școlare este discutabil. Ținând cont că foarte puțini elevi din clasă au ales mai departe acest profil, probabil că materia ar fi trebuit să se axeze pe utilizare. Deși eu aș fi vrut să conteze mai mult partea de programare și algoritmi.

Palmaresul la Olimpiadele elevilor

Cristi Grozea: calificare la Olimpiada Națională de Informatică în 1990 (eram în clasa a XII-a).

Adrian Sandor: De mai multe ori m-am clasat pe locul întâi la faza județeană dar, deoarece m-am calificat și la matematică, nu am participat la națională, cele două olimpiade fiind organizate, aproape întotdeauna, în același timp. În clasele a VII-a și a VIII-a am participat la concursurile dintre cluburile elevilor, la care am luat locul întâi pe țară (în taberele de la Năvodari).

Bogdan Stănescu: Doar locul I la Olimpiada Județeană de Informatică, când eram în clasa a XI-a.

Acum, la clasele de informatică sunt doar 4 ore/săptămână. Vi se pare suficient? Ce ar trebui să se predea?

Cristi Grozea: Nu știu, părerea mea este că ar trebui insistat pe algoritmi, plus un singur limbaj de programare. Mi-a folosit foarte mult un trimestru de scheme logice în clasa a IX-a.

Liviu Panait Alexandru: Pe vremea când eram elev erau 8 ore pe săptămână, aproximativ 3 materii diferite și aveam și laboratoare (4 ore/săptămână). Pentru majoritatea elevilor era suficient, iar cine dorea să studieze suplimentar, evident nu era împiedicat.

Date biografice

Cristi Grozea născut în august 1972, București
În prezent este preparator la Facultatea de Matematică, secția Informatică, Universitatea București.
e-mail: chrisg@phobos.cs.unibuc.ro.

Liviu Panait Alexandru născut la 24 februarie 1977, București; în prezent este student anul IV
e-mail: liviutzu@yahoo.com.

Adrian Sandor născut în septembrie, 1978 la Onești; în prezent este student anul II
e-mail: adisan@phobos.cs.unibuc.ro.

Bogdan Stănescu născut în iunie 1977, în prezent este student anul IV
e-mail: bogst@phobos.cs.unibuc.ro.

Adrian Sandor: Nu cred că va fi suficient... Oricum, ar fi bine dacă măcar în aceste 4 ore s-ar face informatică, dar din câte am auzit (cel puțin la liceul unde am studiat eu) profesorii de informatică își petreceau mare parte din timp cu cititul catalogului, motivarea unor absențe și dojenirea elevilor care se jucau prea mult și/sau prea zgomotos.

Bogdan Stănescu: Este insuficient. Informatica necesită mult timp pentru învățare, dacă este făcută serios, iar la clasele de profil ar trebui să fie făcută serios.

Cum s-a făcut uniformizarea în anul I între absolvenții claselor de informatică și cei care puneau pentru prima dată mâna pe calculator?

Cristi Grozea: Destul de simplu. Am preferat să nu opresc evoluția celor care știau mult mai mult, așa că nu s-a uniformizat nivelul. Sper că toți au urcat mai multe trepte.

Liviu Panait Alexandru: Uniformizarea s-a făcut cam așa: cine știa puțină informatică nu trebuia să vină la cursul de informatică până prin semestrul al II-lea.

Adrian Sandor: Cred că fiecare s-a descurcat cum a putut. Pentru un necunoscător este practic imposibil să învețe numai din cursuri, seminarii și laboratoare. Are nevoie de un calculator la care să lucreze și în timpul liber, de cineva care să-l ajute și de materiale suplimentare. Asta bineînțeles dacă vrea să învețe. Însă mulți se mulțumesc să copieze, să memoreze câteva lucruri și să învețe doar ceea ce este strict necesar pentru a se descurca.

Bogdan Stănescu: Trebuie avut în vedere că este o facultate de matematică. Majoritatea celor care nu au avut nimic comun cu informatica înainte de anul I își aleg din anul II un profil de matematică. Este evident că uniformizarea nu se poate face deoarece unele licee de informatică sunt foarte bune.

Urmăriți Gazeta de Informatică? De cât timp și cât de des? V-a folosit la ceva? Ce credeți că ar trebui să conțină?

Cristi Grozea: Încă nu. Dar, după conținut, pare să merite a fi citită.

Liviu Panait Alexandru: Am urmărit primele numere și m-au cam dezamăgit. Asta se întâmpla acum câțiva ani buni. De curând am văzut noua față a revistei și mi-a plăcut. Cred că direcția aleasă este foarte bună (pe mine m-a convins că merită citită).

Adrian Sandor: Da, mai ales de la începutul lui 1998 când am văzut din întâmplare un exemplar din "seria nouă" fiindcă mai aveam acasă și niște numere vechi (pe care nu le-am studiat cu atenție pentru că eram prea mic când le-am văzut prima oară, iar mai târziu mi s-au părut interesante, dar nu mi-au plăcut în mod deosebit). De atunci am citit cu atenție fiecare număr, în prezent fiind abonat. Am găsit multe lucruri interesante, unele mi-au folosit și practic. Revista ar trebui să continue în forma actuală, eventual cu mai multe prezentări de algoritmi, structuri de date, "API"-uri, formate de fișiere și prezentări de programe. De asemenea, ar fi bine ca articolele să fie mai mari și mai clare, mai detaliate (în general am observat că subiectele sunt tratate succint și unele chiar superficial).

Bogdan Stănescu: Nu o urmăresc regulat deoarece am aflat de ea de puțin timp. Orice articol dintr-o revistă de informatică este cu adevărat util unei categorii limitate de cititori.

Merită să organizezi/participi la un concurs gen ACM, dat fiind timpul limitat? Ce mod de abordare diferit presupune față de Olimpiadele elevilor? Este mai interesant/atractiv sau este numai "altfel"?

Cristi Grozea: *Evaluarea automată* constituie un mare avantaj. Nu știu dacă s-a introdus și pentru elevi.

Liviu Panait Alexandru: Întotdeauna merită să participi la un concurs, deoarece câștigi experiență. Problema timpului necesar pregătirii este foarte dificilă, deoarece majoritatea profesorilor nu înțeleg cât timp îți poate lua antrenamentul pentru concurs.

Avantaje:
- înveți să lucrezi în echipă
- este recunoscut peste hotare (au auzit la toate universitățile din lume de acest concurs)
- pe orice CV arată bine un locul I la un concurs internațional

Adrian Sandor: Nu știu cum se organizează un concurs ACM, dar sunt sigur că este o experiență foarte interesantă. Să participi - cu atât mai mult: este singurul concurs "de prestigiu" pentru studenți, de care am auzit eu și mă atrag perspectivele pe care le oferă: pe de o parte, participarea la faza finală înseamnă vizitarea unei țări străine (anul acesta Olanda), iar premiile înseamnă bani (cuvânt la care studenții nu rămân indiferenți...); apoi mai este perspectiva unei angajări la o firmă "puternică" de soft din străinătate (de exemplu IBM).

Modul de desfășurare a concursului mi se pare mai interesant și mai atractiv decât olimpiadele, în primul rând datorită lucrului în echipă (este necesară o bună coordonare pentru a putea lucra în paralel, trei concurenți având un singur calculator). Nu trebuie să aștepți până la sfârșitul concursului pentru a afla dacă o problemă este rezolvată corect sau nu; există șansa ca un program care nu a primit "OK" să fie corectat și predat din nou. De asemenea este interesantă și departajarea în funcție de momentul în care o rezolvare a fost predată, ceea ce impune concurenților să recunoască problemele ușoare și să le rezolve rapid.

Bogdan Stănescu: Merită să participi la astfel de concursuri pentru satisfacția rezultatelor, dar trebuie să te antrenezi. Este un concurs diferit de olimpiade prin lucrul exclusiv în echipă și prin modul de corectare al programelor: un program se punctează numai dacă funcționează corect pentru toate testele.

V-au folosit la concurs cunoștințele asimilate în facultate?

Cristi Grozea: Da.

Liviu Panait Alexandru: Da, materia studiată la mai multe cursuri a ajutat la rezolvarea unor probleme mai dificile.

Adrian Sandor: Nu. La facultate se pune mai mult accent pe teorie și apoi eu de abia începeam anul II când am participat la concurs și în anul I nu am învățat nimic nou și folositor în același timp.

Bogdan Stănescu: Da, cursurile de algoritmi, în special cele din anul II.

De ce ați ales Facultatea de Matematică și nu Politehnica sau A.S.E.-ul? Vă pare rău acum?

Cristi Grozea: Datorită lui Tudor Sorin. Nu îmi pare rău.

Liviu Panait Alexandru: De A.S.E. nici nu s-a pus problema (secția de cibernetică este extrem de slabă la capitolul informatică). De la Poli am făcut rost de cursuri și nu prea mi-a plăcut ce am văzut. În principiu nu îmi pare rău, deoarece am învățat să privim lucrurile într-un mod foarte abstract și să raționăm altfel decât un absolvent al Politehnicii.

Adrian Sandor: Am venit la Universitate deoarece îmi place matematica; nu mă interesează fizica și electronica (cel puțin nu la nivel de facultate) și nici economia; în afară de asta, Facultatea de Matematică durează doar 4 ani. După ce am început anul I, cu cât am aflat mai multe informații despre diverse facultăți am fost cu atât mai convins (și mai bucuros) că am făcut alegerea cea bună.

Bogdan Stănescu: Am ales Universitatea deoarece îmi place mai mult matematica decât fizica. Dacă aș fi din nou în situația de a alege prima facultate, aș alege tot Matematica.

Ce ar trebui să se facă în facultate și nu se face?

Cristi Grozea: Îmi era mult mai clar când eram student.

Liviu Panait Alexandru: Cred că ar trebui să li se acorde mai multă importanță profesorilor de informatică, altfel cei mai buni vor continua să renunțe la a preda în Facultatea de Matematică. De asemenea, mi-ar plăcea să existe mai multe materii de informatică în locul celor de matematică.

Adrian Sandor: Ar trebui să se pună un accent mult mai mare pe practică și mai ales pe domeniile actuale ale informaticii. Dar acest lucru este aproape imposibil datorită dotării extrem de slabe a facultății și datorită faptului că sunt foarte puține cadre didactice competente, iar printre acestea sunt foarte puțini tineri. De ce? Pentru că nu sunt bani și tinerii deștepți pleacă. Cred că s-a discutat suficient pe această temă în revistele de specialitate (inclusiv în GInfo). Ca să dau câteva exemple, ar trebui să se învețe măcar până la sfârșitul anului II limbajul C++ și limbajul de asamblare; ar trebui să se folosească cel puțin un mediu de programare în Windows.

Bogdan Stănescu: Ar trebui să existe un mod oficial de apreciere a profesorilor după "calitatea cursurilor" pe care le țin, nu numai după activitatea de cercetare.

Lucrați în prezent? În ce domeniu al informaticii?

Cristi Grozea: Programare. CGI-uri în Perl și portare Unix->Windows.

Liviu Panait Alexandru: Am lucrat, dar am renunțat pentru că nu prea mai aveam timp de școală. Cred că o să am timp suficient după ce termin facultatea.

Adrian Sandor: În prezent nu sunt angajat, dar aș putea spune că lucrez singur. În general mă interesează toate domeniile informaticii, dar mai ales grafica (cu tot ce înseamnă ea), muzica, limbajele (formale, naturale), construcția sistemelor de operare și a compilatoarelor, inteligența artificială, algoritmi pentru jocuri, programarea în Internet, algoritmi de compresie și criptare și, în general, aplicarea matematicii în informatică (ordinea este oarecum arbitrară). În unele din aceste domenii sunt doar începător.

Bogdan Stănescu: În prezent nu lucrez. Am lucrat programare Windows API și MFC.

Ce intenționați să faceți în viitor?

Cristi Grozea: Greu de spus. Deocamdată sunt preparator și antrenez echipa Universității.

Adrian Sandor: Aș vrea să mă angajez ca programator, de preferat în străinătate (eventual după sau în același timp cu studiile de doctorat) sau chiar în țară (în funcție de salariu). Oricum, nu intenționez să rămân în străinătate; visul meu este să am firma mea de soft în România, care să devină cunoscută în lume.

Bogdan Stănescu: Să lucrez la o firmă, dacă se poate cu sediul în țară. Iar dacă voi rămâne în țară, mi-ar plăcea să rămân asistent la o universitate de informatică.

ZERO-polinoame

Mihai Scorțaru

Pare-se, există o cale lungă de la definiția polinomului, învățat/neînvățat în școala generală și până la programele care implementează eficient operațiile cu polinoame.

Matematică

Unul dintre cele mai interesante domenii ale matematicii, cu numeroase aplicații în diferite științe (fizică, chimie, informatică etc.) este *studiul polinoamelor.*

Prima "întâlnire" cu polinoamele (în cl. a-VIIa)

În clasa a VII-a, într-una dintre orele de matematică, atenția elevilor va fi captată de către profesor, în momentul în care acesta va prezenta noul subiect al lecției: *Astăzi vom vorbi despre monoame și polinoame!* Majoritatea elevilor, contrariați de acest ciudat subiect, își vor întrerupe activitățile obișnuite (jocuri de tip X-O) și vor privi interogativ spre profesor, așteptând ca acesta, printr-o scurtă frază, să le dezvăluie misterul problemei. Noua "imagine" a clasei - elevii interesați de subiect - va bucura profesorul. Prima definiție (*Monomul este un produs între o constantă și una sau mai multe variabile.*) va adânci și mai mult misterul subiectului orei de matematică; lecția continuă până la o nouă definiție: *Polinomul este o sumă de monoame*, care va descuraja elevii și aceștia, derutați, își vor relua activitățile obișnuite.

A doua "întâlnire" cu polinoamele (în cl. a-IXa)

În clasa a IX-a, la ora de matematică, un alt profesor va încerca să prezinte clasei un nou subiect: *Astăzi vom vorbi despre șiruri.* În amintirea elevilor va apărea scenariul din clasa a VII-a care nu i-a impresionat deloc. La o întrebare a profesorului: *Cine știe ce este un polinom?* cineva, care probabil nu a avut partener la X-O în clasa a VII-a, răspunde: *O sumă de monoame!!!* Zâmbind, profesorul îl aprobă: *Da, dar să dăm acum o definiție riguroasă!* Colegii celui care a avut ideea de a răspunde la întrebare îi aruncă priviri amenințătoare, privind speriați spre profesorul care dă o nouă definiție: *Un polinom este un șir ai cărui termeni sunt nuli începând de la o anumită poziție*. Prima oară despre polinoame se va încheia fără ca elevii să fie pe deplin lămuriți în ceea ce privește noile noțiuni.

Elevii claselor de informatică vor întâlni acest subiect și în orele de specialitate, când vor încerca să simuleze operațiile cu polinoame. Profesorul va face o scurtă recapitulare a noțiunilor, prezentându-le mai întâi din punct de vedere matematic, iar apoi informatic: *un polinom va fi reprezentat ca un șir în care elementul cu indicele* 0 *va fi coeficientul lui* X^0, *elementul cu indicele* 1 *va fi coeficientul lui* X^1 *etc.*

O "întâlnire" specială cu polinoamele (în Gazeta de Informatică)

Aici vom întâlni o nouă abordare a acestui subiect: *Reprezentarea clasică a polinoamelor nu este convenabilă în cazul în care acestea au mulți coeficienți nuli.* Dacă partenerul de X-O lipsește, vă recomandăm să citiți cele ce urmează!!!

Definiție și reprezentare

Vom numi ***zero-polinom*** *un polinom pentru care numărul coeficienților nuli este mult mai mare decât numărul celor nenuli.*

Vom reprezenta un zero-polinom ca o listă ordonată în care fiecare element va conține *gradul unui monom nenul* și *valoarea coeficientului.* Această listă va fi ordonată descrescător după gradul monoamelor. Pentru a ușura implementarea diferitelor operații cu polinoame, vom folosi o listă ordonată cu santinelă la capăt. Dacă doriți să economisiți spațiul de memorie pe care îl va ocupa această santinelă, vă propunem să încercați implementarea operațiilor care vor urma, folosind o listă fără santinele.

Interfața

În continuare vă vom prezenta un ***unit*** care conține proceduri pentru implementarea operațiilor uzuale cu polinoame: *citirea* și *scrierea unui polinom*, *atribuirea valorii unui polinom altui polinom*, *adunarea*, *scăderea*, *înmulțirea* și *împărțirea a două polinoame*. Interfața acestui ***unit*** arată astfel:

Secțiunea interface a unit-ului

```
interface
type Reference=^Element;
     Element=record
               value:Real;        { valoarea }
               degree:Longint;    { gradul   }
               next:Reference
          { legatura spre următorul element }
             end;
     Polynominal=record
                   first,last:Reference
                 end;
{ tipul Polynominal este un articol care    }
{ contine cate o referinta la primul,       }
{ respectiv ultimul element al polinomului  }
procedure ReadPolynominal(var P:Polynominal);
                                   { Citirea  }
procedure WritePolynominal(P:Polynominal;
                           poz:Byte);
                                   { Afisarea }
procedure AssignPolynominal(P:Polynominal;
                        var Q:Polynominal);
          { Atribuirea valorii unui polinom }
          { altui polinom                   }
procedure AddPolynominals(P,Q:Polynominal;
                          var R:Polynominal);
               { Adunarea a doua polinoame }
procedure SubtractPolynominals
        (P,Q:Polynominal; var R:Polynominal);
                { Scaderea a doua polinoame }
procedure MultiplyPolynominals
        (P,Q:Polynominal; var R:Polynominal);
             { Inmultirea a doua polinoame }
procedure DividePolynominals(P,Q:Polynominal;
                        var R,S:Polynominal);
             { Impartirea a doua polinoame }
```

Implementarea

În paginile următoare vom prezenta implementarea procedurilor care au apărut în interfața *unit*-ului. Nu vom explica modalitățile de adăugare a unui element într-o listă ordonată, cei nefamiliarizați putând consulta diferite manuale care tratează acest subiect [1].

Citirea unui zero-polinom

Pentru a inițializa un zero-polinom vom citi *gradul* și *valoarea* pentru fiecare element, în ordinea descrescătoare a gradelor. Valorile trebuie să fie nenule. În cazul în care se vor introduce valori greșite, se va cere introducerea unei noi valori. Vom considera acțiunea terminată în cazul în care se va introduce un grad negativ. Elementele vor fi întotdeauna adăugate la capătul listei ordonate, acest lucru fiind posibil datorită modului în care am făcut citirea. Procedura este următoarea:

Procedura de citire a polinomului

```
procedure ReadPolynominal(var P:Polynominal);
var value:Real;
    degree,lastdegree:Longint;
    r:Reference;
begin
  with p do
  begin
    Writeln;
    Write('Please enter the value and the ');
    Write('degree of the terms of the ');
    Witeln('polynominal:');
    Write('Enter the degree of the term:');
    Readln(degree);
    lastdegree:=degree;
    New(first);              { initializarea }
    first^.next:=nil;        { santinelei    }
    last:=first;
    while degree>=0 do
    begin
      repeat
        Write('Enter the value of the term:');
        Readln(value);
        if value = 0
        then Writeln('Wrong value:');
      until value<>0;
         { Nu se accepta valoarea 0          }
         { Adaugarea unui element în lista }
      New(r);
      r^.value:=value;
      r^.degree:=degree;
      r^.next:=nil;
      last^.next:=r;          last:=r;
      repeat
        Write('Enter the degree of the ');
        Write(term:');
        Readln(degree);
        if degree>=lastdegree
        then Writeln('Wrong degree!!!';
      until degree<lastdegree;
     { Gradul trebuie sa fie mai mic decat }
     { gradul elementului anterior         }
      lastdegree:=degree
    end
  end
end;
```

Propunem ca exercițiu implementarea acestei proceduri în cazul în care elementele polinomului nu sunt introduse în ordinea descrescătoare a gradelor. (*Indicație:* se va construi o listă ordonată descrescător după grade.)

Afișarea unui zero-polinom

Vom afișa un polinom într-un format destul de obișnuit; de exemplu polinomul $-5X^4+X^2-X+2$ va apărea pe ecran astfel: `-5X^4+X^2-X+2`.

Pentru a realiza acest lucru este necesar să tratăm separat cazurile în care gradul este 1 sau 0, cazul în care valoarea absolută a coeficientului este 0 și cazul primului element pentru care nu va trebui să afișăm semnul '+', iar semnul '-' nu va trebui separat de valoarea coeficientului printr-un spațiu. În cazul polinomului nul vom afișa doar un 0. (Un polinom, reprezentat în modul ales de noi, este *nul* dacă valoarea elementului imediat următor santinelei va fi **nil**.)

Luând în considerare aceste observații, ajungem la următoarea procedură de afișare a unui polinom "creat" cu procedura ReadPolynominal:

Procedura de afișare a polinomului

```
procedure WritePolynominal(P:Polynominal;
                           poz:Byte);
 { poz indica numarul de zecimale care      }
 { vor fi afisate pentru fiecare coeficient }
var r:Reference;
begin
  with p do
  begin
    if first^.next<>nil
    then
    begin
      r:=first^.next;
      while r<>nil do
      begin
        if r<>first^.next
        then Write(' ');
        { vom pune spatiu numai daca       }
        { nu suntem la primul termen (**)  }
        if r^.value<0
        then Write('-')
       { punem semnul '-' in cazul in care }
       { numărul este negativ              }
        else
          if r<>first^.next
          then Write('+');
{ punem semnul '+' numai daca numarul       }
{ este pozitiv si nu suntem la primul termen }
        if r<>first^.next
        then Write(' ');
        { se pune spatiu numai daca        }
        { nu suntem la primul termen (**)  }
        if (Abs(r^.value)<>1) or
                              (r^.degree=0)
        then Write(Abs(r^.value):0:poz);
       { se scrie numarul numai daca       }
       { valoarea sa absoluta nu este 1    }
       { sau daca suntem la ultimul element }
        if r^.degree>0
        then Write('X');
                  { se afiseaza un 'X' numai }
                  { daca gradul nu  este 0   }
        if r^.degree>1
        then Write('^',r^.degree);
         { se afiseaza puterea, numai daca }
         { gradul este mai mare decat 1    }
        r:=r^.next
           { se trece la elementul urmator }
      end
    end
    else
      Write('0')     { Pentru polinomul nul }
  end
end;
```

În rândurile marcate cu (**) se arată că spațiul nu trebuie afișat în cazul primului termen, pentru a putea respecta modelul de afișare cerut. Dacă nu am fi pus aceste condiții, polinomul $-5X^4+X^2-X+2$ ar fi arătat astfel:

```
-5X^4+X^2-X+2.
```

S-a pus o condiție și pentru semnul '+' pentru a se evita o afișare de genul:

```
+5X^4+X^2-X+2
```

pentru polinomul $5X^4+X^2-X+2$.

Parametrul poz a fost introdus pentru a permite alegerea numărului de cifre care vor fi afișate după punctul zecimal.

Atribuiri pentru zero-polinoame

Pentru a atribui unei variabile de tip polinom o altă variabilă de tip polinom, vom copia elementele din lista unui polinom, în lista celuilalt polinom. Deoarece această procedură este simplă, vom trece direct la reproducerea codului sursă corespunzător:

Procedura de atribuire

```
procedure AssignPolynominal(P:Polynominal;
                            var Q:Polynominal);
var r,s:Reference;
begin
  with q do
  begin
    New(first);              { initializarea }
    first^.next:=nil;        { santinelei    }
    last:=first;
    r:=p.first^.next;
    while r<>nil do
    begin
          { se copiaza un element din        }
          { lista primului polinom in lista  }
          { celui de-al doilea polinom       }
      New(s);
      s^.value:=r^.value;
      s^.degree:=r^.degree;
      s^.next:=nil;
      last^.next:=s;
```

```
      last:=s;
      r:=r^.next
         { se trece la elementul urmator }
    end
  end
end;
```

Nu se poate face o atribuire directă de tipul `P:=Q` deoarece câmpurile articolelor `P` și `Q` nu conțin decât referințe la elementele polinomului, iar o atribuire de genul acesta nu ar face decât ca cele două variabile să indice că elementele polinomului se află în aceeași zonă de memorie. Pentru a face o dublură "reală" a primului polinom, trebuie să procedăm așa cum s-a arătat mai sus.

Adunarea a două zero-polinoame

Datorită modului de reprezentare a polinoamelor, algoritmul de obținere a polinomului sumă seamănă foarte mult cu *algoritmul de interclasare a două liste ordonate*. Să presupunem, pentru început, că toate elementele care apar în cele două polinoame au gradele distincte. În acest caz polinomul sumă va fi obținut într-adevăr prin interclasarea celor două liste în care sunt păstrate elementele celor două polinoame. Dacă vor exista elemente cu același grad, acestea vor fi tratate separat, în polinomul sumă fiind introdus un element care va avea gradul celor două elemente și valoarea egală cu suma valorilor celor două elemente. Trebuie să verificăm dacă această sumă este 0, caz în care nu vom mai adăuga nici un element. Nu insistăm asupra procedeului de interclasare a două liste; cititorii care doresc explicații suplimentare pot consulta diferite cărți în care este prezentat acest subiect([1]). Procedura pe care o propunem pentru adunarea a două polinoame este următoarea:

Procedura de adunare a două polinoame

```
procedure AddPolynominals(P,Q:Polynominal;
var R:Polynominal);
var s,t,u:Reference;
    degree:Longint;
    value:Real;
begin
  with r do
  begin
    New(first);              { initializare }
    first^.next:=nil;        { santinela    }
    last:=first;
    s:=p.first^.next;
    t:=q.first^.next;
    while (s<>nil) and (t<>nil) do
    begin
      if s^.degree>t^.degree
      then
      begin
    { adaugam un element din prima lista si }
    { trecem la elementul ei urmator        }
      degree:=s^.degree;
      value:=s^.value;
      s:=s^.next
    end
    else
      if s^.degree<t^.degree
      then
      begin
    { adaugam un element din a doua lista }
    { si trecem la elementul ei următor   }
      degree:=t^.degree;
      value:=t^.value;
      t:=t^.next
    end
      else
      begin
    { adaugam suma elementelor din cele }
    { doua liste si trecem la elementul }
    { urmator în ambele liste           }
      degree:=s^.degree;
      value:=t^.value + s^.value;
      s:=s^.next;
      t:=t^.next
    end;
    if value<>0
    then  { nu putem adauga valoarea 0 }
    begin
      New(u);
      u^.degree:=degree;
      u^.value:=value;
      u^.next:=nil;
      last^.next:=u;
      last:=u
    end
  end;
{ adaugam elementele ramase in prima lista }
  while s<>nil do
  begin
    New(u);
    u^.degree:=s^.degree;
    u^.value:=s^.value;
    u^.next:=nil;
    last^.next:=u; last:=u;
    s:=s^.next
  end;
{ adaugam elementele ramase in a doua lista }
  while t<>nil do
  begin
    New(u);
    u^.degree:=t^.degree;
    u^.value:=t^.value;
    u^.next:=nil;
    last^.next:=u; last:=u; t:=t^.next
  end
 end
end;
```

Este evident că nu pot rămâne elemente în ambele liste, deci corpul cel puțin uneia din cele două instrucțiuni **while** din finalul procedurii nu va fi executat nici măcar o dată. Există și posibilitatea să determinăm în care din cele două liste au rămas elemente și să adăugăm numai elementele din acea listă, ceea ce ar duce la eliminarea unuia din cele două cicluri. Propunem ca exercițiu implementarea procedurii de adunare a două polinoame în acest mod.

Scăderea a două zero-polinoame

Procedura pentru scăderea a două polinoame nu diferă în mod esențial de procedura de adunare, singura diferență fiind faptul că elementele celui de-al doilea polinom trebuie considerate cu semn schimbat. Lăsăm pe seama cititorilor scrierea acestei proceduri.

O altă posibilitate de implementare a scăderii a două polinoame ar fi schimbarea semnelor elementelor din cel de-al doilea polinom și efectuarea unei adunări. Pentru aceasta ar trebui implementată o procedură care să determine opusul unui polinom. Polinomul obținut ar fi trebuit și el memorat, iar aceasta ar fi dus la creșterea spațiului de memorie necesar pentru ca programul să se excute în bune condiții.

Înmulțirea a două zero-polinoame

Pentru a înmulți două polinoame vom lua pe rând monoamele din primul polinom și le vom înmulți cu monoamele din cel de-al doilea polinom. Vom obține astfel mai multe polinoame (câte unul pentru fiecare monom conținut de primul polinom) pe care va trebui să le adunăm. Pentru a scurta timpul de execuție am efectuat aceste operații simultan, așa cum se vede în procedura următoare:

Procedura de înmulțire a două polinoame

```
procedure MultiplyPolynominals
       (P,Q:Polynominal; var R:Polynominal);
var s,t,u,v:Reference;
    value:Real;
    degree:Longint;
begin
  with r do
  begin
    New(first);                  { initializare }
    first^.next:=nil;            { santinela    }
    last:=first;
    s:=p.first^.next;
    while s<>nil do
    begin
      t:=q.first^.next;
      u:=first^.next;
      while t<>nil do
      begin
        { se calculeaza valoarea si gradul }
        { elementului din produs           }
        value:=s^.value*t^.value;
        degree:=s^.degree+t^.degree;
        while (u^.degree>degree) and
                                (u<>nil) do
          u:=u^.next;
          { se determina pozitia in care }
          { trebuie introdus elementul   }
        if u=nil
        then
        begin
          { elementul trebuie adăugat pe }
          { ultima pozitie a listei      }
          New(v);
          v^.degree:=degree;
          v^.value:=value;
          v^.next:=nil;
          last^.next:=v;
          last:=v
        end
        else
          if u^.degree=degree
          then
            u^.value:=u^.value+value
    { exista deja un monom cu acest grad (**)}
          else
          begin
        { se insereaza un element in lista }
            New(v);
            v^:=u^;
            u^.degree:=degree;
            u^.value:=value;
            u^.next:=v
          end;
        t:=t^.next
          { se trece la elementul urmator  }
          { din al doilea polinom          }
      end;
      s:=s^.next
          { se trece la elementul următor  }
          { din primul polinom             }
    end;
    u:=first^.next;
    while u<> nil do
      if u^.value=0
      then
      begin
    { se sterg elementele care au valoarea 0 }
        v:=u^.next;
        u^:=u^.next^;
        Dispose(v)
      end
      else u:=u^.next
  end
end;
```

În final trebuie eliminate elementele care au valoarea 0, deoarece în momentul efectuării adunării marcate cu (**) nu am testat dacă rezultatul este 0. Propunem ca exercițiu implementarea procedurii de înmulțire a două polinoame, ținând cont de această observație. Datorită unui rezultat matematic destul de ușor de demonstrat, nu a trebuit să testăm, în momentul ștergerii, dacă suntem la ultimul element, deoarece acesta este întotdeauna nenul.

Împărțirea a două zero-polinoame

Pentru a împărți două polinoame vom proceda într-o manieră "convențională", implementând algoritmul de împărțire învățat (mai mult sau mai puțin) de elevii claselor a VII-a. Vom inițializa un polinom S cu valoarea deîmpărțitului și, la fiecare pas, vom găsi un monom care va fi adăugat câtului. Acest monom va avea gradul egal cu diferența dintre gradul lui S și gradul împărțitorului și valoarea egală cu câtul dintre valorile primelor elemente din cele două polinoame. Vom înmulți acest monom cu împărțitorul și polinomul rezultat îl vom scădea din S. Algoritmul se va termina în momentul în care gradul lui S va fi mai mic decât gradul împărțitorului. În final în S vom obține câtul împărțirii. Procedura corespunzătoare este:

Procedura de împărțire a două polinoame

```
Procedure DividePolynominals(P,Q:Polynominal;
                             var R,S:Polynominal);
var t,v:Polynominal;
    u:Reference;
    value:Real;
    degree:Longint;
begin
  AssignPolynominal(p,s);
              { initializarea polinomului S }
  with r do
  begin
    New(first);                { initializare }
    first^.next:=nil;          { santinela    }
    last:=first;
    while (s.first^.next <> nil) and
          (s.first^.next^.degree>=
           q.first^.next^.degree) do
    begin
    { determinarea monomului care va fi     }
    { adaugat in lista elementelor catului }
      value:=s.first^.next^.value/
                        q.first^.next^.value;
      degree:=s.first^.next^.degree-
                        q.first^.next^.degree;
                    { adaugarea elementului }
      New(u);
      u^.degree:=degree;
      u^.value:=value;
      u^.next:=nil; last^.next:=u; last:=u;
      AssignPolynominal(q,t);
      u:=t.first^.next;
    { inmultirea dintre monom si impartitor }
      while u<>nil do
      begin
        u^.value:=u^.value*value;
        u^.degree:=u^.degree+degree;
        u:=u^.next
      end;
    { scaderea din S a polinomului obtinut }
    { in urma inmultirii precedente        }
      SubtractPolynominals(s,t,v);
      AssignPolynominal(v,s)              {(**)}
    end
  end
end;
```

Înaintea liniei marcate cu (**) ar fi trebuit eliberat spațiul de memorie ocupat de vechiul polinom S pentru ca acest spațiu să poată fi refolosit. Pentru aceasta ar mai trebui implementată o procedură de ștergere DeletePoly, a cărei implementare o lăsăm pe seama cititorilor.

Încheiere

Dacă partenerul de X-O nu s-a întors între timp, înseamnă că ați citit încă un articol "inutil" despre polinoame; informațiile articolului "pot fi de folos" pentru foarte puțini dintre dumneavoastră. Totuși, după cum probabil nu ați observat, în prima parte nu am spus nimic despre ce face, de obicei, un profesor universitar când trebuie să țină cursul despre polinoame. S-ar putea ca unii dintre dumneavoastră, să-i facă o surpriză profesorului dând dovada deținerii unui bagaj de cunoștințe destul de bogat în acest domeniu într-un moment nu tocmai potrivit. De exemplu, este posibil ca scopurile didactice urmărite de către profesor să nu fie "convergente" cu dorința voastră firească de a semnala că sunteți as în "de-ale programării".

Atenție!!! Surprizele pot fi și neplăcute! Pot avea consecințe imprevizibile asupra notei finale pe care o veți obține la materia respectivă!...

A sosit timpul să vă urăm succes la X-O. În final, am dori să ne cerem scuze față de profesorii care nu au ținut lecția despre polinoame în modul descris de noi. (Ne-am "jucat" cu subiectul și evident, nu ne-am fi putut "juca" dacă nu am fi avut profesori de la care să învățăm multe.) Le adresăm pe această cale cele mai sincere felicitări.

Bibliografie

1. Rancea, D. - *Limbajul Turbo Pascal*, Editura Libris - 1994
2. Horowitz, E., Sahni, S., Mehta, D. - *Fundamentals in Data Structures in C++*, Computer Science Press - 1995

Mihai Scorțaru este student la Universitatea Tehnică Cluj; poate fi contactat prin e-mail la adresa: scort@licj.soroscj.ro

CRIPTARE cu chei publice

Victor-Valeriu Patriciu

Vlad și Gabi doresc să dialogheze, transmițând unul celuilalt mesaje "top secret" pe Internet. Să vedem cum procedează...

În evoluția criptografiei moderne, în anul 1976, ***Whitfield Diffie*** și ***Martin Hellman***, de la Universitatea Stanford din California, au descoperit un principiu diferit de cifrare. Ei au pus bazele ***criptografiei asimetrice cu chei publice***. În locul unei singure chei secrete, criptografia asimetrică folosește două chei diferite: una pentru cifrare, alta pentru descifrare. Deoarece este imposibilă deducerea unei chei din cealaltă, una din chei este făcută publică, fiind pusă la îndemâna oricui dorește să transmită un mesaj cifrat. Doar destinatarul, care deține cea de-a doua cheie, poate descifra și utiliza mesajul. Tehnica cheilor publice poate fi folosită și pentru autentificarea mesajelor prin semnătură digitală, fapt care i-a sporit popularitatea.

În cripto-sistemele având chei publice, un utilizator, de exemplu *Vlad*, va deține:
- o *transformare de cifrare publică (cheia publică)*, notată E_{Vlad}, care poate fi memorată într-un fișier public;
- o *transformare de descifrare secretă (cheie privată sau secretă)*, notată D_{Vlad}, ce este imposibil să fie obținută din E_{Vlad}.

Cheia de descifrare (secretă) este derivată din cheia de cifrare (publică) printr-o *transformare greu inversabilă (one-way)*. În sistemele cu chei publice, secretizarea și autentificarea sunt realizate prin transformări distincte. Iată cum sunt folosite sistemele cu chei publice:

- Să presupunem că utilizatorul *Vlad* dorește să transmită un mesaj M unui alt utilizator, *Gabi*. *Vlad* cunoaște dintr-un fișier (server) transformarea (cheia) publică E_{Gabi} și poate transmite pe M lui *Gabi* sub forma criptată C:

$$C=E_{Gabi}(M).$$

Se asigură astfel ***funcția de confidențialitate***.

La recepție *Gabi* va descifra criptograma C utilizând transformarea (cheia) sa secretă D_{Gabi}, cunoscută doar de ea:

$$D_{Gabi}(C)=D_{Gabi}(E_{Gabi}(M))=M.$$

Schema nu asigură autentificare, deoarece orice utilizator are acces la cheia publică E_{Gabi} a lui *Gabi* și îi poate trimite mesaje false M' sub forma $C'=E_{Gabi}(M')$.

- Pentru ***autentificare***, se pune problema recunoașterii adevăratului emițător. Pentru aceasta se aplică lui M transformarea secretă D_{Vlad} a lui *Vlad*. Acesta va calcula și apoi va trimite lui *Gabi* criptograma C:

$$C=D_{Vlad}(M).$$

La recepție, *Gabi* va aplica transformarea publică, E_{Vlad} a lui *Vlad*:

$$E_{Vlad}(C)=E_{Vlad}(D_{Vlad}(M))=M.$$

Numai *Vlad* cunoaște și poate aplica transformarea D_{Vlad}. Observăm că nu este asigurată și confidențialitatea mesajului, întrucât este posibil ca M să fie decriptat de orice user ce are acces la fișierul (serverul) cu chei publice, aplicând în exemplul nostru transformarea publică E_{Vlad}.

- Pentru a se realiza simultan ***confidențialitatea*** și ***autentificarea***, emițătorul de mesaj *Vlad* va aplica mai întâi transformarea sa secretă D_{Vlad} pentru autentificarea mesajului M. Apoi *Vlad* va cifra rezultatul, utilizând transformarea publică a lui *Gabi*, E_{Gabi} și va emite către receptor criptograma:

$$C=E_{Gabi}(D_{Vlad}(M)).$$

Receptorul *Gabi* îl obține pe M aplicând la început propria-i funcție de descifrare D_{Gabi}, iar apoi transformarea publică a lui *Vlad* E_{Vlad}, cea care furnizează autentificarea:

$$\begin{aligned} E_{Vlad}(D_{Gabi}(C)) &= E_{Vlad}(D_{Gabi}(E_{Gabi}(D_{Vlad}(M)))) \\ &= E_{Vlad}(D_{Vlad}(M)) \\ &= M. \end{aligned}$$

În figura 1 se arată modalitatea de asigurare a confidențialității (secretul) unui mesaj, folosind criptarea cu chei publice. Pentru aceasta, mesajul este cifrat cu cheia publică a destinatarului, operație ce poate fi făcută de către orice persoană care poate accesa fișierul cu chei publice. După cifrare mesajul nu va mai putea fi descifrat decât de către destinatar, singurul care posedă cheia secretă (privată), pereche a celei publice. Când doi utilizatori vor să schimbe mesaje securizate, fiecare transmite o cheie celuilalt, denumită ***cheie publică*** și o păstrează pe cealaltă, denumită ***cheie privată***. Deoarece mesajul criptat cu cheia

publică poate fi decriptat doar cu cea privată, mesajele pot fi transmise prin rețea fără a exista riscul ca un intrigant să citească datele transmise.

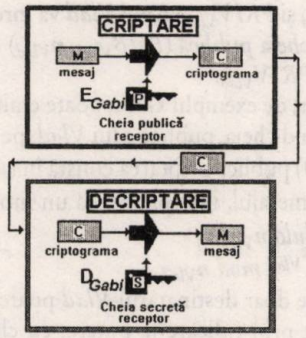


Figura 1 Sisteme de cifrare cu chei publice

În practica aplicațiilor Internet, cifrurile cu chei publice sunt folosite în general pentru următoarele 2 servicii:

- ***semnătura digitală*** asociată mesajelor, pentru autentificarea conținutului și a emițătorului;
- ***cifrarea și distribuția cheilor simetrice*** (de tip DES, IDEA etc.), folosite în secretizarea mesajelor, operație numită ***anvelopare a cheii.***

Semnătura digitală rezolvă atât problema autentificării emițătorului, cât și pe cea a autentificării conținutului mesajului însuși. În exemplul din figura 2 se prezintă utilizarea sistemelor cu chei publice pentru ***semnare digitală.*** Dacă, de exemplu, *Gabi* primește un mesaj semnat de *Vlad*, semnătura lui *Vlad* trebuie să satisfacă următoarele *proprietăți*:

- *Gabi* să fie capabilă să valideze semnătura lui *Vlad*;
- semnătura lui *Vlad* să fie imposibil de falsificat, chiar pentru *Gabi*;
- în cazul în care *Vlad* nu recunoaște semnarea mesajului, trebuie să existe un "judecător" care să poată rezolva disputa dintre *Vlad* și *Gabi*.

Sistemele de autentificare cu chei publice permit o implementare simplă a semnăturilor digitale. Cheia privată, deoarece este deținută doar de *Vlad*, poate servi ca element de calcul a semnăturii digitale pentru *Vlad*. *Gabi*, receptorul mesajului *M* semnat, poate verifica atât autenticitatea emițătorului, cât și pe cea a datelor. Validarea semnăturii se poate face de către *Gabi*, cu cheia publică pereche, a lui *Vlad*.

În practică, o semnatură digitală se realizează folosind un sistem criptografic cu chei publice și o ***funcție H de dispersie*** (*hash*). Funcția de dispersie este folosită pentru a calcula o *valoare rezumat* (*digest*) care depinde de toți biții mesajului ce va fi semnat. Pentru o astfel de funcție, (de exemplu pentru cea cunoscută sub numele de MD5, proiectată de *Rivest*), trebuie să fie computațional imposibil să se construiască 2 mesaje distincte care să aibă aceeași valoare rezumat; acest lucru face din rezumat o "amprentă" a mesajului. Utilizatorul emițător, în loc să semneze mesajul *M*, care poate avea sute sau mii de kilobytes, semnează rezumatul (de 128 de biți), prin cifrare cu cheia sa privată. Semnătura poate fi validată apoi de receptor folosind cheia publică a emițătorului, așa cum se poate vedea în figura 2. La recepție, se calculează din nou rezumatul mesajului primit, se descifrează semnătura primită cu cheia publică a emițătorului și apoi se compară cele 2 rezumate. Dacă ele sunt identice, semnătura este validă.

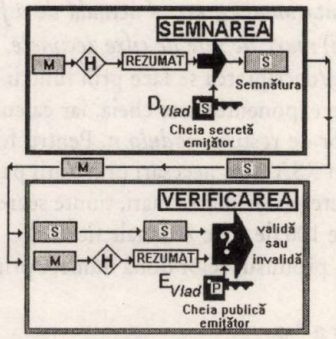


Figura 2 Sisteme de semnătură digitală

Criptografia are ca scop principal confidențialitatea mesajelor. Acestea sunt criptate cu o cheie de cifru simetric, de exemplu IDEA sau DES, generată aleator. Pentru a se putea distribui receptorului această cheie, ea este criptată, la rândul ei, cu cheia publică a primitorului mesajului, obținându-se așa-numita ***anvelopă digitală*** a mesajului. *Învelișul* (*anvelopa*) digital(ă) este trimis receptorului împreună cu mesajul criptat. După primirea datelor, receptorul decriptează învelișul digital, folosind cheia sa privată, pentru a obține cheia simetrică generată aleator de către emițător, pe care apoi o folosește la decriptarea mesajului original.

Algoritmii de criptare cu cheie publică prezintă o cripto-complexitate foarte mare, bazându-se în general pe operații cu întregi foarte mari (sute de cifre zecimale sau mii de biți). Dintre problemele matematice dificile care stau la baza acestor cifruri, amintim *factorizarea unor numere întregi mari*, *calculul logaritmilor în câmp finit* sau *problema rucsacului*. Implementările existente de sisteme de criptare cu chei publice folosesc, cel mai adesea, următorii algoritmi criptografici:

- pentru rezumat: ***MD2***, ***MD5*** (*Mesage Digest* creat de *Ronald Rivest*), ***SHA*** (*Secure Hash Algoritm*, creat de Institutul de Standarde al SUA - *NIST* pentru standardul de semnătură DSA);
- pentru criptare/semnătură: ***RSA*** (creat de *Rivest*, *Shamir* și *Adleman*), *El Gamal*, ***DSA*** (*Digital Signature Algorithm*, creat de *NIST* ca standard de semnătură digitală), ***Curbe Eliptice*** și ***Metoda Rucsacului*** a lui *Merkle-Hellman*.

Vom prezenta două dintre cele mai cunoscute sisteme criptografice cu chei publice.

Cifrul RSA

RSA reprezintă cel mai folosit sistem cu chei publice și semnături digitale. A fost realizat de trei cercetători - *Rivest, Shamir, Adleman* - de la ***MIT*** (*Massachusetts Institute of Technology*) și are o mare susținere atât în mediul guvernamental, cât și în cel comercial. Sub diferite forme de implementare, prin software sau prin dispozitive hardware speciale, RSA este astăzi recunoscută ca cea mai sigură metodă de cifrare și autentificare disponibilă comercial. Ideea pe care își bazează puterile RSA constă în *imposibilitatea computațională practică actuală de a factoriza numere (întregi) mari, de sute de cifre zecimale.*

Criptarea/decriptarea se face prin funcții de tip exponențial, unde exponentul este cheia, iar calculele se fac în inelul claselor de resturi *modulo n*. Pentru folosirea unui cripto-sistem RSA sunt necesari următorii parametrii:

- două numere prime foarte mari, ținute secrete, p și q (de exemplu de 100 de cifre zecimale fiecare);
- *modulul n*, produsul celor două numere prime mari, făcut public:

  $n = p * q$.

Securitatea RSA depinde de dificultatea factorizării lui n în p și q. Rivest, Shamir și Adleman recomandă folosirea unor numere prime p și q de 100 cifre zecimale, adică a unui n de 200 de cifre zecimale, ceea ce cere pentru factorizare mai multe milioane de ani calculator. Ultimele recomandări făcute de firma ***RSA Data Security***, deținătoarea patentului RSA, pentru alegerea dimensiunii cheilor de cifrare RSA (de fapt a modulului n) sunt următoarele:

- în cazul unor *aplicații cu cerințe de securitate pe termen scurt:* 768 biți, dimensiune suficientă în aplicații pentru utilizatori individuali. Se precizează că informația este sigură 1 an;
- în cazul unor *aplicații cu cerințe de securitate pe termen mediu*: 1024 biți, dimensiune suficientă în aplicații comerciale. Se apreciază că informația este sigură 3 ani;
- în cazul unor *aplicații cu cerințe de securitate pe termen lung:* 2048 biți, dimensiune considerată suficientă pentru aplicații cu cerințe de securitate sporită, la nivel strategic militar sau guvernamental. Se apreciază că, în acest caz, informația este sigură 20 de ani.

Plecând de la parametrii specificați se calculează ***indicatorul lui Euler*** $(n)=(p-1)*(q-1)$, care este imposibil de determinat de un atacator, întrucât nu se cunosc factorii primi ai lui n, p și q.

Baza teoretică a lui RSA este *generalizarea lui Euler a Teoremei lui Fermat* care stabilește că:

$M^{\varphi(n)} = 1 \ mod \ n$.

Această proprietate implică necesitatea ca cele 2 chei, ***PUB*** și ***PRIV*** să fie inverse multiplicativ *modulo* $\varphi(n)$. Cele 2 chei cu care operează RSA sunt:

- ***PRIV - cheia privată,*** aleasă ca un întreg relativ prim cu $\varphi(n)$, în intervalul $[max(p,q)+1, n-1]$;
- ***PUB - cheia publică,*** un întreg calculat ca invers multiplicativ modulo $\varphi(n)$:

  $PUB * PRIV = 1 \ mod \ \varphi(n)$.

Criptarea unui mesaj

Se generează cu un program special pentru fiecare utilizator, modulul n și cheile *PUB* și *PRIV*. În cazul exemplului pe care-l vom analiza, user-ul *Vlad* obține modulul n_{Vlad} și cheile PUB_{Vlad} și $PRIV_{Vlad}$. Apoi *Vlad* va înregistra într-un fișier public, ***cheia publică*** (PUB_{Vlad}, n_{Vlad}) și ține secretă ***cheia privată*** $PRIV_{Vlad}$.

Un alt user, de exemplu *Gabi*, poate emite un *mesaj M criptat*, utilizând cheia publică a lui *Vlad*, pe care o găsește în fișierul (BD) public. Criptarea constă în ridicarea biților ce reprezintă mesajul, interpretați ca un întreg, la puterea PUB_{Vlad}, *modulo* n_{Vlad}:

$C = M^{PUB_{Vlad}} \ mod \ n_{Vlad}$.

La recepție doar destinatarul *Vlad* poate obține mesajul în clar, tot prin ridicare la putere, cu cheia sa secretă $PRIV_{Vlad}$:

$$M = C^{PRIV_{Vlad}} \ mod \ n_{Vlad}$$
$$= (M^{PUB_{Vlad}})^{PRIV_{Vlad}} \ mod \ n_{Vlad}$$
$$= M^{PUB_{Vlad} * PRIV_{Vlad}} \ mod \ n_{Vlad} = M,$$

conform teoremei lui Euler, deoarece:

$PUB*PRIV=1 \ mod \ \varphi(n) = (k* \ \varphi(n) +1)$.

Semnarea unui mesaj

User-ul *Vlad* va putea semna un mesaj M, calculând cu cheia sa secretă $PRIV_{Vlad}$:

$S = M^{PRIV_{Vlad}} \ mod \ n_{Vlad}$.

Un alt user, de exemplu *Gabi*, poate autentifica acest mesaj, utilizând cheia publică PUB_{Vlad} a lui *Vlad* și calculând, tot prin ridicare la putere, mesajul M din semnătura S recepționată:

$M= S^{PUB_{Vlad}} \ mod \ n_{Vlad}$.

Semnătura se consideră a fi validă în cazul în care mesajul astfel calculat coincide cu cel trimis direct prin rețea.

În practica semnării digitale, se utilizează un rezumat al mesajului M, $H(M)$, calculat cu o ***funție de hash,*** H; acesta este mai scurt decât M, având de obicei 128 de biți.

Vom ilustra folosirea algoritmului RSA printr-un exemplu numeric.

Fie p=19 și q=43 cele două numere prime secrete alese pentru userul *Vlad*. Valoarea *modulului* n_{Vlad} (produsul celor două numere) este $n_{Vlad} = 43*19 = 817$. Indicatorul lui Euler calculat este:

$(n)=(p-1)*(q-1)=42*18=756$.

Se alege, de exemplu, cheia privată a lui *Vlad*, $PRIV_{Vlad}$=625. Se calculează apoi cheia publică $PUB_{Vlad} = (PRIV_{Vlad})^{-1} \ mod \ 756 = (625)^{-1} \ mod \ 756 = 227$. Se va publica pe server sau într-un fișier cheia publică PUB_{Vlad} și n_{Vlad} și se va ține secretă valoarea $PRIV_{Vlad}$.

Pentru a cripta mesajul:

M= "astazi ma duc la scoala"

îl vom sparge mai întâi în blocuri mai mici, în cazul nostru de câte o cifră:

M_1 = "a" = 97, M_2 = "S" = 115 etc.

Primul caracter va fi criptat ca:

$97^{227} \ mod \ 817 = 451$, $115^{227} \ mod \ 817 = 628$ etc.

Restul caracterelor sunt prelucrate prin exponențiere la fel, obținându-se criptograma:

C=451 628 679 451 122 528 409 212 451 409 662 268 310 409 352 451 409 628 310 701 451 352 451

Pentru a decripta mesajul, se va ridica fiecare bloc criptat la puterea 625:

451^{625} *mod* 817 = 97 = "*a*" etc.

Cifrul MERKLE-HELLMAN (MH)

Metoda de ***cifrare cu cheie publică MH*** este bazată pe cunoscuta ***problemă a rucsacului,*** care constă în a determina într-o mulțime de numere întregi, o submulțime de o sumă dată. ***Merkle*** și ***Hellman*** propun o metodă a cărei securitate depinde de dificultatea rezolvării următoarei probleme:

Fiind dat un întreg pozitiv C și un vector $A=(a_1,a_2,...,a_n)$ de întregi pozitivi, să se găsească un subset al lui A a cărui sumă să fie C. Cu alte cuvinte este necesar să se determine un vector binar $M=(m_1,...,m_n)$, astfel încât $C=A \times M$ sau $C=\sum a_i \cdot m_i$.

Intuitiv, problema este următoarea:

Cunoscând greutatea unui rucsac închis și greutățile mai multor obiecte care s-ar putea afla în interiorul său, să se determine setul de obiecte aflate în rucsac, fără a-l deschide.

Această problemă este *NP completă*. Cel mai bun algoritm cunoscut pentru rezolvarea ei - în cazul în care dimensiunea rucsacului este *n* - cere $O(2^{n/2})$ timp și $O(2^{n/2})$ memorie. Există totuși o clasă specială de probleme de un asemenea tip, numită ***clasa problemelor cu rucsac simplu*** ce pot fi rezolvate într-un timp liniar. Într-un rucsac simplu elementele a_i (i=1, ..., n) sunt în *relația de dominanță*, adică:

$a_i > \sum a_j$, unde j=1, i.

Relația de dominanță simplifică ***soluția rucsacului,*** după următorul algoritm:

```
procedura RUCSAC-SIMPLU (C,A) este
  pentru i=n,1 execută
    dacă C >= a_i
    atunci m_i=1 altfel m_i=0
  atribuie C=C - Σ a_i*m_i
  sfârșit_pentru
  dacă C = 0
  atunci "Soluția este în M"
  altfel "Nu există soluție"
sfârșit_procedura
```

În proiectarea unui astfel de criptosistem trebuie convertit *rucsacul simplu* într-un ***rucsac cu trapă aditivă*** (***trapdoor knapsack***), care este greu de rezolvat. Mai întâi se selectează un vector *rucsac simplu* $A'=(a'_1, a'_2, ..., a'_n)$. Acesta permite o soluție simplă a problemei, $C'=A'M$. Apoi se alege un întreg *n*, astfel ca:

$n > 2\, a'_n > \sum a'_i$, unde i=1,...,n.

În continuare se alege un alt întreg *w*, astfel ca *cmmdc(n,w)*=1 și se calculează inversul lui *w mod n*. În final vectorul *A'* este transformat într-un vector *rucsac greu*:

$A=w \times A' \bmod n$, $a_i = w \times a_i' \bmod n$.

Acum, rezolvarea problemei $C=A \times M$ este dificilă, dacă nu se dispune de o informație "trapă" (inversul lui *w* și *n*), cu ajutorul căreia calculul se simplifică:

$$\begin{aligned} C' &= w^{-1} C \times mod\ n \\ &= w^{-1} A \times M \times mod\ n \\ &= w^{-1} (w \times A') \times M \bmod n \\ &= A' M \bmod n \\ &= A' M. \end{aligned}$$

Transformarea de cifrare PUB (publică) folosește *cheia publică* care este vectorul *rucsac greu A*. Se obține criptograma $C = PUB(M) = A \times M$.

Transformarea de descifrare *PRIV* folosește *cheia secretă* (A', n, w^{-1}), calculând pe baza funcției *rucsac simplu*:

$PRIV(C)=RUCSAC\_SIMPLU(w^{-1}C \bmod n, A')=M$.

Pentru a se înțelege mai clar algoritmul *Merkle-Hellman*, să considerăm următorul exemplu numeric:

Fie $A'=(1,3,5,10)$ vectorul rucsac simplu, n=20 și se alege w=7. Ca urmare, inversul multiplicativ al lui *w modulo* 20 este w'^{-1}=3. Acest *rucsac simplu* este transformat într-unul *cu trapă aditivă A* astfel:

$A= (7*1 \bmod 20, 7*3 \bmod 20, 7*5 \bmod 20, 7*10 \bmod 20)$
$= (7, 1, 15, 10)$.

Se consideră următorul mesaj *M*, care va fi cifrat:

M=13, în binar $M=(1, 1, 0, 1)$.

Criptograma *C* este obținută cu ajutorul vectorului cu trapă *A*:

$C=PUB(M)=A;\ M=7+1+10=18$.

Ca urmare, avem $C=(1, 0, 0, 1, 0)$.

La descifrare se obține mesajul clar, cu vectorul simplu *A'* care este secret:

$M=PRIV(C)=PRIV(18)=$
$=RUCSAC\_SIMPLU(3*18 \bmod 20, A')=$
$=RUCSAC\_SIMPLU(14, A')=$
$=(1, 1, 0, 1)=$
$=13$

Merkle și *Hellman* au sugerat alegerea în proiectarea unui cripto-sistem bazat pe acest algoritm, a unui rucsac cu n=100 elemente (sau chiar mai mare), ceea ce face ca cifrul să fie computațional inatacabil.

În numerele viitoare ne vom opri asupra unor aplicații ale sistemelor criptografice, simetrice sau cu chei publice, în serviciile Internet.

Domnul Victor-Valeriu Patriciu este cadru didactic la Academia Tehnică Militară. Poate fi contactat prin e-mail la adresa: vip@mta.ro

Pagini WEB cu HTML 4.0

Mihai Scorțaru

Până acum majoritatea elementelor prezentate în acest serial se concentrau asupra modului în care putem transmite informații către navigatorii în Internet, singura excepție constituind-o link-urile e-mail prezentate în primul episod. HTML nu este "o stradă cu sens unic"! Paginile Web pot fi folosite și pentru a obține informații de la persoanele care le vizitează. Acest episod vă prezintă modul în care pot fi create formularele Web, precum și modalități de prelucrare a datelor transmise cu ajutorul acestora.

Elementele unei pagini Web care permit vizitatorilor să introducă informații care să fie transmise creatorului (posesorului) paginii Web se numesc *formulare*. Toate formularele sunt asemănătoare, și există doar două metode prin care browserul poate trimite informații către serverul care găzduiește pagina Web. Dacă informațiile trimise de browser sunt incluse în URL-ul destinație, metoda folosită este denumită *get*. Dacă URL-ul destinație este unul normal (nu conține elemente suplimentare) metoda este denumită *post*. Modalitățile de transmitere ale datelor vor fi prezentate într-o întreagă secțiune din acest episod.

Tag-ul <FORM>

Formularele HTML sunt delimitate de comanda **<FORM>** a cărei sintaxă este:

```
<FORM ACTION = url METHOD = metodă>
  <!-- conținut formular -->
</FORM>
```

URL-ul indică fișierul sau aplicația aflată (de obicei) pe serverul care urmează să prelucreze informațiile, iar *metoda* indică modalitatea prin care vor fi transmise informațiile (`get` sau `post`).

Între *opening tag*-ul și *closing tag*-ul comenzii **<FORM>** pot fi conținute orice informații HTML de formatare, de grafică, link-uri etc., dar și o mulțime de elemente noi, specifice formularelor.

Elemente ale unui formular

Majoritatea câmpurilor unui formular pot fi specificate cu ajutorul comenzii **<INPUT>** care poate avea o mulțime de atribute, excepție făcând meniurile derulante și casetele de text care conțin mai multe linii. Comenzile corespunzătoare ultimelor două elemente sunt **<SELECT>**, respectiv **<TEXTAREA>**.

Parametrii comenzii **<INPUT>** sunt următorii:

- **type** indică tipul câmpului de intrare;
- **name** indică numele simbolic al valorii câmpului;
- **value** precizează conținutul prestabilit al câmpului (dacă acesta este un câmp de tip `text`);
- **checked** precizează faptul că respectivul câmp este inițial validat (pentru butoane radio sau casete de validare);
- **size** precizează numărul caracterelor casetei afișate;
- **maxlength** precizează numărul maxim acceptat de caractere (pentru câmpuri de tip text).

Cele mai uzuale valori pe care le poate lua parametrul **type** sunt: `text`, `password`, `checkbox`, `hidden`, `file`, `radio`, `submit`, `image` și `reset`.

Text este tipul prestabilit și creează o casetă de tip text care va conține o singură linie.

Password (*parolă*) creează o casetă de tip text, dar în locul oricărui caracter introdus de către utilizator este afișat caracterul '*', din motive de securitate.

Checkbox (*casetă de validare*) creează o casetă de validare; atributul `CHECKED` specifică faptul că această casetă va fi marcată în mod prestabilit; cu ajutorul opțiunii `VALUE` puteți specifica textul asociat casetei de validare.

Hidden (*ascuns*) permite transmiterea datelor fără ca utilizatorul să le poată vedea pe ecran.

File (*fișier*) permite utilizatorului să transmită serverului un fișier. Va apare un buton de tip `Browse` care va permite selectarea fișierului dorit.

Radio afișează un buton de interblocare; diferite astfel de butoane care au același nume sunt grupate în mod automat, nefiind permisă selectarea mai multor butoane din același grup.

Submit (*transmite*) determină apariția unui buton care, atunci când este apăsat, transmite conținutul întregului formular serverului aflat la distanță.

Image este un câmp aproape identic cu `submit`, în locul butonului putând fi specificată o imagine grafică.

Reset determină apariția unui buton cu ajutorul căruia utilizatorul poate șterge conținutul tuturor câmpurilor formularului, valorile acestora devenind cele prestabilite.

Cu ajutorul tag-ului **<SELECT>** poate fi creat un meniu derulant de opțiuni. Între opening tag-ul și closing tag-ul acestei comenzi trebuie să apară un nou tag, **<OPTION>**, care va permite definirea fiecărei opțiuni a meniului. Parametrii tag-ului **<SELECT>** sunt:

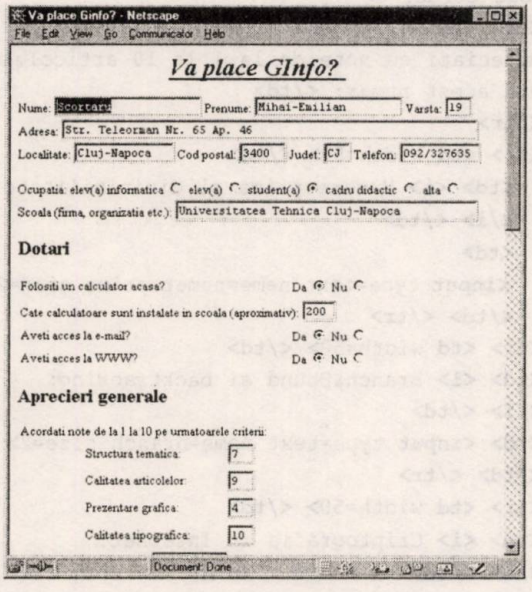
- **name** indică *numele meniului derulant*;
- **size** indică *numărul opțiunilor afișate în același timp*;
- **multiple** indică faptul că *utilizatorii pot selecta mai mult decât o opțiune.*

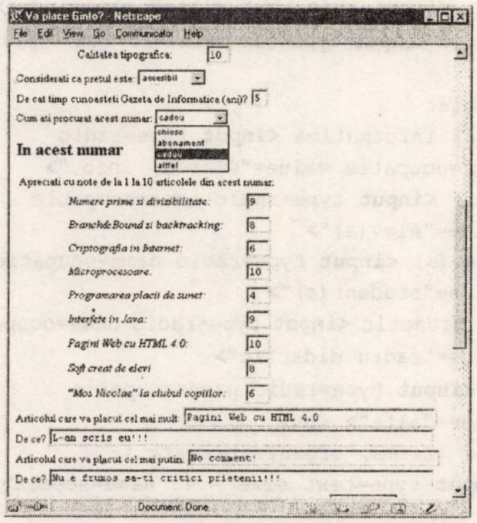
Pentru a specifica faptul că o anumită opțiune este prestabilită trebuie să folosiți atributul `SELECTED` al tag-ului **<option>** corespunzător opțiunii respective.

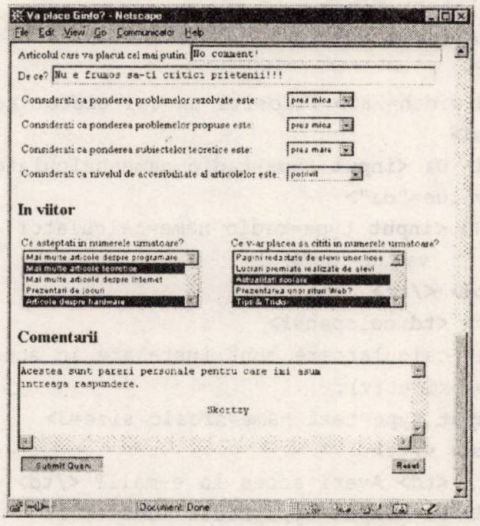
Comanda **<TEXTAREA>** vă permite să inserați în formularul dumneavoastră o casetă de tip `text` care să conțină mai multe linii. Parametrii acestei comenzi sunt `name`, `rows` și `col`, indicând *numele*, *numărul de linii*, respectiv *numărul de coloane* al casetei.

Primele formulare

Pentru a exemplifica modul de utilizare al câmpurilor unui formular vă prezentăm modul în care poate fi creat un formular asemănător celui din ultima pagină a revistei. Iată cum ar arăta acest formular într-o pagină Web:







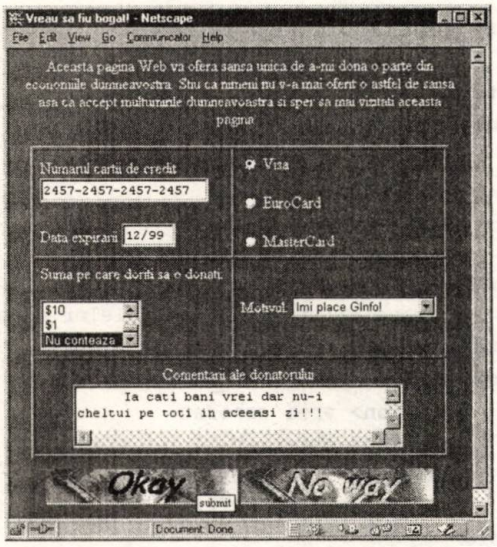
Fișierul HTML corespunzător este:

```
<html> <head>
<title> Va place Ginfo? </title>
</head>
<body>
<center> <h1><i><u>
  Va place GInfo?
</h1></i></u> </center>
<form action=
  http://www.ginfo.ro/skortzy/ginfo.cgi
  method=get>
Nume: <input type=text size=20 name=nume>
Prenume: <input type=text size=20 name=prenume>
Varsta: <input type=text size=2 name=varsta>
<br>
Adresa: <input type=text size=60 name=varsta>
<br>
Localitate: <input type=text size=13
  name=localitate maxlength=25>
Cod postal: <input type=text size=5
  name=codpostal>
```

```
Judet: <input type=text size=2 name=judet>
Telefon: <input type=text size=10 name=telefon>
<p>
Ocupatia:
elev(a) informatica <input type=radio
  name=ocupatie value="elev(a) info.">
elev(a) <input type=radio name=ocupatie
  value="elev(a)">
student(a) <input type=radio name=ocupatie
  value="student(a)">
cadru didactic <input type=radio name=ocupatie
  value="cadru didactic">
alta <input type=radio name=ocupatie
  value="alta"> <br>
Scoala (firma, organizatia etc.):
  <input type=text size = 43 name=scoala>

<h2> Dotari </h2>
<table border=0>
  <tr>
  <td width=280> Folositi un calculator acasa
  </td>
  <td> Da <input type=radio name=calculator
    value="da">
    Nu <input type=radio name=calculator
        value="nu">
  </td> </tr>
  <tr> <td colspan=2>
  Cate calculatoare sunt instalate in scoala
  (aproximativ):
  <input type=text name=nrcalc size=3>
  </td> </tr>
  <tr> <td> Aveti acces la e-mail? </td>
  <td> Da <input type=radio name=email
    value="da">
   Nu <input type=radio name=email value="nu">
      </td> </tr>
  <tr> <td> Aveti acces la WWW? </td>
 <td> Da <input type=radio name=WWW value="da">
  Nu <input type=radio name=WWW value="nu">
</td> </tr>
</table>

<h2> Aprecieri generale </h2>
<table border=0>
  <tr> <td colspan=3>
  Acordati note de la 1 la 10 pe urmatoarele
criterii: </td>
  </tr>
    <td width=50> </td>
    <td> Structura tematica: </td>
    <td>
      <input type=text name=structura size=2>
    </td> </tr>
  <tr>
    <td width=50> </td>
    <td> Calitatea articolelor: </td>
    <td>
     <input type=text name=calitate_articole
      size=2>
    </td> </tr>
  <tr>
    <td width=50> </td>
    <td> Prezentare grafica: </td>
    <td>
     <input type=text name=prezentare size=2>
    </td> </tr>
  <tr>
    <td size=50> </td>
    <td> Calitatea tipografica: </td>
    <td>
      <input type=text
       name=calitate_tipografica size=2>
    </td> </tr>
</table>
Considerati ca pretul este:
<select name=pret>
<option> prea mare
<option> accesibil
<option> prea mic
</select> <br>
De cat timp cunoasteti Gazeta de Informatica
(ani)?
<input type=text name=timp size=1> <br>
Cum ati procurat acest numar:
<select name=procurat>
<option> chiosc
<option> abonament
<option> cadou
<option> altfel </select>
<h2> In acest numar </h2>

<table border=0>
  <tr> <td colspan=3>
  Apreciati cu note de la 1 la 10 articolele
  din acest numar: </td>
  </tr>
  <tr> <td width=50> </td>
    <td> <i> Numere prime si divizibilitate:
    </i> </td>
    <td>
     <input type=text name=numereprime size=2>
    </td> </tr>
  <tr> <td width=50> </td>
  <td> <i> Branch&Bound si backtracking:
  </i> </td>
  <td> <input type=text name=branch size=2>
  </td> </tr>
  <tr> <td width=50> </td>
  <td> <i> Criptografia in Internet:
  </i> </td>
  <td>
```

```
  <input type=text name=criptografie size=2>
</td> </tr>
<tr> <td width=50> </td>
<td> <i> Microprocesoare: </i> </td>
<td> <input type=text name=microprocesoare
      size=2>
</td> </tr>
<tr> <td width=50> </td>
  <td> <i> Programarea placii de sunet:
</i> </td>
<td> <input type=text name=sunet size=2>
</td> </tr>
<tr> <td width=50> </td>
  <td> <i> Interfete in Java: </i> </td>
  <td> <input type=text name=Java size=2>
</td> </tr>
<tr> <td size=50> </td>
 <td> <i> Pagini Web cu HTML 4.0: </i> </td>
  <td> <input type=text name=HTML size=2>
  </td> </tr>
<tr> <td size=50> </td>
<td> <i> Soft creat de elevi </i> </td>
<td> <input type=text name=soft size=2>
</td> </tr>
<tr> <td size=50> </td>
<td> <i> "Mos Nicolae" la clubul copiilor:
</i> </td>
<td> <input type=text name=MosNicolae size=2>
</td> </tr>
</table>
Articolul care va placut cel mai mult:
<input type=text name=best size=40> <br>
De ce?
<input type=text name=decebest size=62> <br>
Articolul care va placut cel mai putin:
<input type=text name=worst size=40> <br>
De ce?
<input type=text name=deceworst size=62> <br>
<table>
  <tr>
  <td> Considerati ca ponderea
       problemelor rezolvate este:
  </td>
  <td> <select name=pondere_rezolvate>
       <option> prea mica
       <option> potrivita
       <option> prea mare
       </select> </td> </tr>
  <tr> <td> Considerati ca ponderea
            problemelor propuse este:
  </td>
  <td> <select name=pondere_propuse>
       <option> prea mica
       <option> potrivita
       <option> prea mare
       </select> </td> </tr>
  <tr> <td> Considerati ca ponderea
            subiectelor teoretice este:
  </td>
  <td> <select name=pondere_teoretice>
       <option> prea mica
       <option> potrivita
       <option> prea mare
       </select> </td> </tr>
  <tr> <td> Considerati ca nivelul de
       accesibilitate al articolelor este:
  </td>
  <td> <select name=nivel_accesibilitate>
       <option> prea coborat
       <option> potrivit
       <option> prea ridicat
       </select> </td> </tr>
</table>

<h2> In viitor </h2>
<table>
  <tr> <td width=250> Ce asteptati in numerele
                      urmatoare? <br>
  <select multiple name="asteptati">
  <option> Mai multe articole despre programare
  <option> Mai multe articole teoretice
  <option> Mai multe articole despre Internet
  <option> Prezentari de jocuri
  <option> Articole despre hardware
  </select> </td>
  <td width=350> Ce v-ar placea sa cititi in
                 numerele urmatoare? <br>
  <select multiple name="like">
  <option> Pagini redactate de elevii unor licee
  <option> Lucrari premiate realizate de elevi
  <option> Actualitati scolare
  <option> Prezentarea unor situri Web?
  <option> Tips & Tricks
  </select> </td> </tr>
</table>

<h2> Comentarii </h2>
<textarea name=comentarii rows=5 cols=65>
</textarea>
<table>
<tr>
  <td width=425> <input type=submit> </td>
  <td> <input type=reset>
</td>
</tr>
</table>
</form>
</body>
</html>
```

În următoarea imagine puteți vedea un formular care conține o imagine în locul butonului Submit:



Fișierul HTML care generează această imagine este următorul:

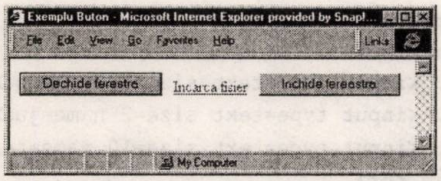
```
<html>
<head> <title> Vreau sa fiu bogat! </title>
</head>
<body bgcolor="#007FAF" text="white">
<center>
Aceasta pagina Web va ofera sansa unica de
a-mi dona o parte din economiile
dumneavostra. Stiu ca nimeni nu v-a mai
oferit o astfel de sansa asa ca accept
multumirile dumneavoastra si sper sa mai
vizitati aceasta pagina.
<form method="post"
action="http://www.ginfo.ro/skortzy/money.cgi">
Selectati aceasta caseta daca doriti sa faceti
o donatie
<input type=checkbox name="donatie" checked>
<p>
<table border=2 cellpadding=5>
  <tr> <td> Numarul cartii de credit <br>
  <input type=text name="numar" size=20> <p>
  Data expirarii
  <input type=text name="data" size=5> </td>
  <td>
  <input type=radio name="plata" value="Visa"
  checked> Visa <p>
  <input type=radio name="plata"
   value="EuroCard"> EuroCard <p>
  <input type=radio name="plata"
   value="MasterCard"> MasterCard
  </td> </tr>
  <tr> <td> Suma pe care doriti sa o donati:
  <p>
  <select name="suma" size=3>
  <option selected> $1.000.000
  <option> $100.000
  <option> $10.000
  <option> $1.000
  <option> $100
  <option> $10
  <option> $1
  <option> Nu conteaza
  </select> </td>
  <td> Motivul:
  <select name="motiv">
  <option> Imi place GInfo!
  <option> Meriti acesti bani!
  <option> Nu merit acesti bani!
  <option> Sunt bogat(a)!
  <option> Sunt fraier(a)!
  <option> Te iubesc!
  <option> Nu stiu!
  </select> </td> </tr>
  <tr> <td colspan=2>
  <center> Comentarii ale donatorului
  <textarea rows=2 cols=40> </textarea>
  </center> </td> </tr>
</table>
<input type="image" src="ok.gif" width=200
 border=0 name="submit">
<a href="money2.html">
<img src="cancel.gif" width=200 border=0> </a>
</form>
</center>
</body>
</html>
```

Butoane

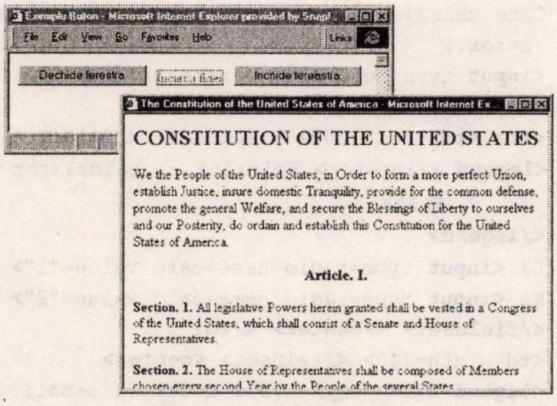
O valoare importantă pe care o poate lua atributul **TYPE** al tag-ului **<INPUT>** este button. Cu ajutorul acestui tip poate fi introdus în pagina Web un buton de uz general. Să considerăm următorul exemplu:

```
<html>
<head> <title> Exemplu Buton </title> </head>
<body>
<form>
  <input type=button value="Dechide ferestra"
  onClick="fereastra=window.open('',
          'fereastra','resizable=yes')">
 
<a href="USA_Constitution.html"
   target='fereastra'>Incarca fisier </a>
 
<input type="button" value="Inchide fereastra"
  onClick="fereastra.close()">
</form>
</body>
</html>
```

La încărcarea imaginii, fereastra browserului va arăta astfel:



Dacă veți efectua un click pe butonul *Deschide fereastra*, va apare o nouă fereastră a browser-ului care nu va conține nimic. Dacă veți efectua un click pe link-ul dintre cele două butoane, în această fereastră va fi încărcat fișierul `USA_Constitution.html`. Iată și o imagine care prezintă cele două ferestre:



Dacă veți efectua un click pe butonul `Inchide fereastra`, noua fereastră va fi închisă.

Am folosit în exemplul anterior secvența de caractere `' '` pentru a introduce un spațiu suplimentar între elementele paginii.

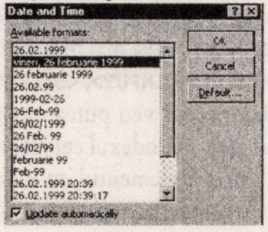
Pentru introducerea butoanelor a fost folosit și un nou tag, **<BUTTON>**, sintaxa acestei comenzi fiind **<BUTTON onClick=...>**. Această nouă comandă oferă aceleași facilități ca și **<INPUT** `type=button onClick=...`**>**.

O altă posibilitate pe care o oferă butoanele este închiderea ferestrei curente. Secvența HTML care creează un buton ce are această funcție poate arăta astfel:

```
<INPUT type=button value="Inchide-ma!"
       onClick="window.close()">
```

Etichete

Într-o aplicație profesională, efectuarea unui click pe un text asociat unui element de intrare duce la modificarea acelui element. Acest text asociat poartă denumirea de ***etichetă*** a elementului. Imaginea următoare prezintă fereastra de dialog oferită de aplicația Microsoft Word '97 pentru introducerea datei și a timpului într-un document.



Pentru a selecta sau a deselecta căsuța din partea inferioară a ferestrei nu este neapărat nevoie să efectuăm click-uri în interiorul pătrățelului corespunzător, un click pe textul asociat având același efect. Acest lucru nu se întâmplă și în cazul formularelor HTML prezentate anterior. Dacă în formularul GInfo veți efectua un click pe cuvântul *cadru didactic*, căsuța corespunzătoare nu va fi selectată; pentru a o selecta va trebui să efectuați un click în interiorul cerculetului asociat.

Specificațiile HTML 4.0 oferă două metode de lucru cu etichetele în cadrul formularelor. O etichetă poate fi specificată prin intermediul parametrului `ID` al tag-ului care introduce elementul, sau cu ajutorul comenzii **<LABEL>** care va încadra tag-ul ce introduce elementul.

Următorul fișier HTML ilustrează modul în care pot fi folosite cele două metode:

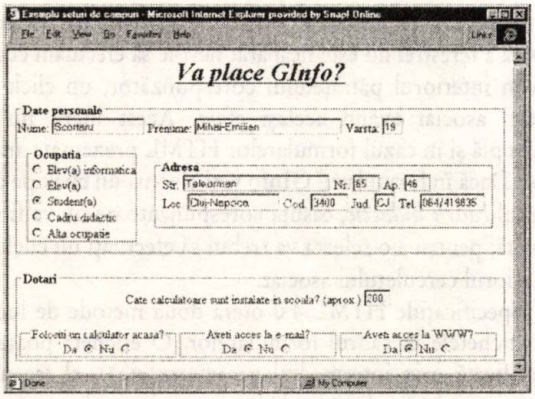
```
<html>
<head>
<title>
  Etichete
</title>
</head>
<body>
<form method=post action=
  "http://www.ginfo.ro/skortzy/etichete.cgi">
Doriti sa invatati HTML?
<input type=radio name=HTML value=da ID=da>
<label for="da"> Da </label>
<label>
  <input type=radio name=HTML value=nu> Nu
</label>
</form>
</body>
</html>
```

Nu vom prezenta o captură a ferestrei browser-ului, deoarece în paginile revistei nu veți putea sesiza faptul că prin efectuarea unui click pe textul asociat, o căsuță ar putea fi selectată. De altfel, chiar dacă acest lucru ar fi fost posibil, Netscape Navigator 4.0 nu asociază textul cu elementul, iar Internet Explorer 4.0 realizează acest lucru numai dacă este folosită prima dintre cele două metode prezentate.

Seturi de câmpuri

Anumite elemente ale unui formular pot fi grupate cu ajutorul tag-ului **<FIELDSET>**. Toate elementele introduse prin comenzi ce apar între opening tag-ul și closing tag-ul comenzii **<FIELDSET>** se consideră că aparțin aceluiași grup. Între opening tag-ul și closing tag-ul acestei comenzi poate apărea comanda **<LEGEND>** care vă permite să atașați o explicație unui anumit grup.

Iată cum ar putea arăta prima parte a formularului GInfo, dacă folosim aceste noi comenzi:



Fișierul HTML care generează această pagină este următorul:

```
<html>
<head>
<title> Exemplu seturi de campuri </title>
</head>
<body>
<center> <h1> <b> <i> <u>
  Va place GInfo?
</b> </i> </u> </h1> </center>
<form method=post action=
  "http://www.ginfo.ro/skortzy/seturi.cgi">
<fieldset>
<legend align='top'> <b> Date personale </b>
</legend>
Nume: <input type=text size=15 name=nume>
Prenume: <input type=text size=25 name=prenume>
Varsta: <input type=text size=2 name=varsta>
<table border=0 cellpadding=10>
  <tr> <td> <fieldset>
  <legend align=top> <b> Ocupatia </b>
  </legend>
  <input type=radio name=ocup value="1">
  Elev(a) informatica<br>
  <input type=radio name=ocupatie value="2">
  Elev(a) <br>
  <input type=radio name=ocupatie value="3">
  Student(a) <br>
  <input type=radio name=ocupatie value="4">
  Cadru didactic <br>
  <input type=radio name=ocupatie value="5">
  Alta ocupatie <br>
  </fieldset>
  </td>
  <td> <fieldset>
 <legend align=top> <b> Adresa </b> </legend>
    Str.
  <input type=text size=25 name=str>
  Nr. <input type=text size=3 name=nr>
  Ap. <input type=text size=3 name=ap> <br>
    Loc.
  <input type=text size=15 maxlength=50
   name=loc>
  Cod. <input type=text size=5 name=cod>
  Jud. <input type=text size=2 name=jud>
  Tel. <input type=text size=10 name=tel>
    <br>
  </fieldset> </td> </tr>
</table>
</fieldset> <p>
<fieldset>
<legend align=top> <b> Dotari </b> </legend>
<table>
  <tr> <td colspan = 3> <center>
  Cate calculatoare sunt instalate in scoala?
  (aprox.)
  <input type=text size=3 name=nrcalc>
  </center> </td> </tr>
  <tr> <td width=200> <fieldset> <center>
  <legend align=top> Folositi un calculator
        acasa?
  </legend>
  Da <input type=radio name=calc value="1">
  Nu <input type=radio name=calc value="2">
  </fieldset> </center> </td>
  <td width=200> <fieldset> <center>
  <legend align=top> Aveti acces la e-mail?
  </legend>
  Da <input type=radio name=email value="1">
  Nu <input type=radio name=email value="2">
  </fieldset> </center> </td>
  <td width=200> <fieldset> <center>
  <legend align=top> Aveti acces la WWW?
  </legend>
  Da <input type=radio name=WWW value="1">
  Nu <input type=radio name=WWW value="2">
  </fieldset> </center> </td> </tr>
</table>
</fieldset>
</form>
</body>
</html>
```

Controlul tastei TAB

Pentru a selecta un element al unui formular puteți fie să efectuați un click pe acel element, fie să vă deplasați de la un element la altul folosind tasta TAB.

Datorită faptului că elementele formularului pot fi așezate în pagină în configurații diferite, este foarte important la care element vă va duce apăsarea tastei TAB. Pentru a asigura controlul asupra acestui fenomen a fost introdus parametrul **TABINDEX** în sintaxa mai multor tag-uri cum ar fi: **<A>**, **<AREA>**, **<OBJECT>**, **<INPUT>**, **<SELECT>**, **<TEXTAREA>** sau **<BUTTON>**. La început veți putea introduce informații pentru elementul care are indexul cel mai mic, iar apăsarea tastei TAB vă va duce la elementul cu indexul imediat următor. Dacă se va ajunge la elementul cu indexul cel mai

mare, prin apăsarea tastei TAB se va trece la cel cu indexul cel mai mic.

Shortcut-uri

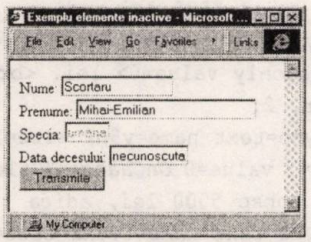
Există un atribut suplimentar care vă permite să simplificați navigarea prin paginile dumneavoastră prin intermediul tastaturii. Acest atribut vă permite să definiți comenzi rapide atașate tastaturii pentru a permite utilizatorilor accesul rapid la anumite câmpuri ale formularului, la un anumit link etc. Atributul poartă denumirea **ACCESKEY** și are ca valoare o anumită tastă. Următoarea secvență HTML permite accesarea câmpului nume cu ajutorul tastei N:

```
<u> N </u> ume <input type=text size=15
                name=nume acceskey='n'>
```

Modul de accesare al câmpului diferă, în funcție de calculatorul folosit. În cazul unui PC va trebui să apăsați simultan tastele ALT și N pentru a accesa elementul respectiv, în timp ce pentru un Macintosh va trebui să apăsați simultan tastele COMMAND și N.

Elemente dezactivate și elemente protejate la scriere

Uneori este nevoie ca unele elemente ale formularelor dumneavoastră să nu poată fi modificate. Pentru a realiza acest lucru specificațiile HTML 4.0 oferă două posibilități: atributele **DISABLED** și **READONLY**. Aceste atribute permit afișarea câmpului, dar nu permit modificarea lui. Singura diferență dintre cele două atribute este aceea că, în timp ce atributul **DISABLED** duce la afișarea câmpurilor astfel încât starea lor inactivă să fie evidentă, câmpurile care au atributul **READONLY** nu se deosebesc, la prima vedere, cu nimic față de celelalte. Iată un astfel de formular:



Fișierul HTML corespunzător este:

```
<html>
<head>
<title> Exemplu elemente inactive </title>
</head>
<body>
<form>
  Nume: <input name=nume size=15 type=text>
  <br>
  Prenume:
  <input name=prenume size=25 type=text> <br>
  Specia: <input name=specie size=5 type=text
        value="umana" disabled> <br>
  Data decesului: <input name=deces size=10
    type=text value="necunoscuta" readonly>
  <br>
<input type=submit value="Transmite">
</form>
</body>
</html>
```

Grupuri de opțiuni

Comanda **<OPTGROUP>** permite utilizatorului să grupeze, în mod logic, mai multe opțiuni care au legătură una cu cealaltă, pentru a simplifica alegerea lor. Acest tag poate fi folosit numai între *opening tag-ul* și *closing tag-ul* comenzii **<SELECT>**. Iată un exemplu:

```
<html>
<head>
<title> Grupuri de optiuni </title> </head>
<body>
<form action=
  "http://www.ginfo.ro/skortzy/grupuri.cgi"
  method="post">
  <p>
  <select name="">
  <optgroup>
  <option> Grupul 1 Optiunea 1
  <option> Grupul 1 Optiunea 2
  <option> Grupul 1 Optiunea 3
  </optgroup>
  <optgroup>
  <option> Grupul 2 Optiunea 1
  <option> Grupul 2 Optiunea 2
  <option> Grupul 2 Optiunea 3
  </optgroup>
  <optgroup>
  <option> Grupul 3 Optiunea 1
  <option> Grupul 3 Optiunea 2
  <option> Grupul 3 Optiunea 3
  </optgroup>
  </select>
</form>
</body> </html>
```

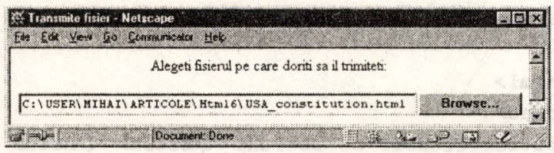
Transmiterea unui fișier

Pentru a permite utilizatorului să transmită un fișier către serverul care găzduiește pagina Web e necesar ca atributul **type** al tag-ului **<input>** să aibă valoarea *file*. În fereastra browserului va apărea o casetă de tip text în care puteți introduce numele fișierului și un buton *Browse* care vă permite deschiderea unei ferestre de tip *browse* cu ajutorul căreia să selectați fișierul dorit. Următorul exemplu ilustrează modul de folosire a valorii *file* pentru atributul **type**.

```
<html>
<head> <title> Transmite fisier </title>
</head>
```

```
<body>
<center>
Alegeti fisierul pe care doriti sa il
trimiteti:
<form>
<input type=file name=fisier size=50>
</form>
</center>
</body>
</html>
```

Fereastra browserului ar putea arăta astfel:



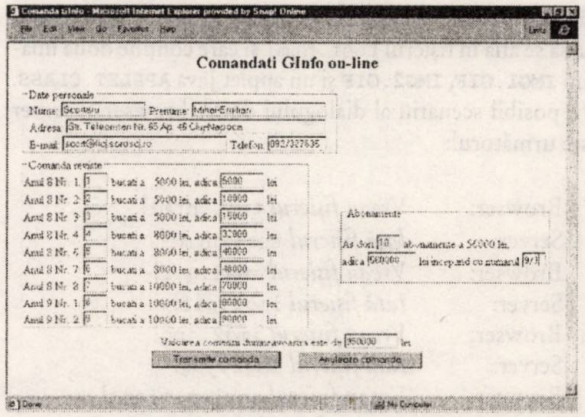
Elemente de programare

Unul dintre cele mai importante domenii în care sunt folosite limbajele de tip script îl reprezintă formularele HTML. De exemplu, în cazul unui formular cu ajutorul căruia se pot comanda mai multe produse, ar fi normal să poată fi calculat costul total al produselor comandate. Pentru a realiza acest lucru (și multe altele) este însă nevoie de scrierea unor script-uri. Să considerăm un formular prin intermediul căruia să poată fi comandată revista GInfo. Clientul ar trebui să introducă, pe lângă datele personale, câte exemplare din fiecare număr al revistei dorește, iar costul total să fie calculat automat. O posibilă sursă HTML care realizează acest lucru este următoarea:

```
<html>
<head> <script language="JavaScript">
  function Calculeaza(){
    t1=0;
    t1+=document.forms[0].tv8n1.
     value=document.forms[0].v8n1.value*5000;
    t1+=document.forms[0].tv8n2.
     value=document.forms[0].v8n2.value*5000;
    t1+=document.forms[0].tv8n3.
     value=document.forms[0].v8n3.value*5000;
    t1+=document.forms[0].tv8n4.
     value=document.forms[0].v8n4.value*8000;
    t1+=document.forms[0].tv8n6.
     value=document.forms[0].v8n6.value*8000;
    t1+=document.forms[0].tv8n7.
     value=document.forms[0].v8n7.value*8000;
    t1+=document.forms[0].tv8n8.
    value=document.forms[0].v8n8.value*10000;
    t1+=document.forms[0].tv9n1.
    value=document.forms[0].v9n1.value*10000;
    t1+=document.forms[0].tv9n2.
    value=document.forms[0].v9n2.value*10000;
    t1+=document.forms[0].tabon.value=
        document.forms[0].abon.value*56000;
    document.forms[0].total.value=t1 }
</script>
<title> Comanda GInfo </title>
</head>
<body>
<center> <h2> Comandati GInfo on-line </h2>
<form action=
 "http://www.ginfo.ro/skortzy/comanda.cgi"
 method=get>
<table>
  <tr> <td width=440>
  <center> <fieldset>
  <legend align=top> Date personale </legend>
   
  Nume: <input type=text name=nume size=15>
  Prenume: <input type=text name=prenume
        size=25> <br>
   
  Adresa: <input type=text name=adresa size=50>
  <br>
   
  E-mail: <input type=text name=email size=31>
  Telefon:
  <input type=text name=telefon size=10>
  <br>
  </center> </fieldset> </td> </tr>
  <tr> <td width=400>
  <fieldset>
  <legend align=top> Comanda reviste </legend>
  Anul 8 Nr. 1:
  <input type=text name=v8n1 size=3
 maxlength=3 value=0 onChange="Calculeaza()">
  bucati a   5000 lei, adica
  <input type=text name=tv8n1 size=7
        readonly value=0> lei. <br>
  Anul 8 Nr. 2:
  <input type=text name=v8n2 size=3
 maxlength=3 value=0 OnChange="Calculeaza()">
  bucati a   5000 lei, adica
  <input type=text name=tv8n2 size=7
        readonly value=0> lei. <br>
  Anul 8 Nr. 3:
  <input type=text name=v8n3 size=3
 maxlength=3 value=0 onChange="Calculeaza()">
  bucati a   5000 lei, adica
  <input type=text name=tv8n3 size=7
        readonly value=0> lei. <br>
  Anul 8 Nr. 4:
  <input type=text name=v8n4 size=3
 maxlength=3 value=0 onChange="Calculeaza()">
  bucati a   8000 lei, adica
  <input type=text name=tv8n4 size=7
        readonly value=0> lei. <br>
  Anul 8 Nr. 6:
  <input type=text name=v8n6 size=3
```

```
  maxlength=3 value=0 onChange="Calculeaza()">
  bucati a   8000 lei, adica
  <input type=text name=tv8n6 size=7
         readonly value=0> lei. <br>
  Anul 8 Nr. 7:
  <input type=text name=v8n7 size=3
  maxlength=3 value=0 onChange="Calculeaza()">
  bucati a   8000 lei, adica
  <input type=text name=tv8n7 size=7
  readonly value=0> lei. <br>
  Anul 8 Nr. 8:
  <input type=text name=v8n8 size=3
  maxlength=3 value=0 onChange="Calculeaza()">
  bucati a 10000 lei, adica
  <input type=text name=tv8n8 size=7
  readonly value=0> lei. <br>
  Anul 9 Nr. 1:
  <input type=text name=v9n1 size=3
  maxlength=3 value=0 onChange="Calculeaza()">
  bucati a 10000 lei, adica
  <input type=text name=tv9n1 size=7
         readonly value=0> lei. <br>
  Anul 9 Nr. 2:
  <input type=text name=v9n2 size=3
  maxlength=3 value=0 onChange="Calculeaza()">
  bucati a 10000 lei, adica
  <input type=text name=tv9n2 size=7
  readonly value=0> lei. <br>
  </fieldeset> </td>
  <td> <fieldset>
  <legend> Abonamente </legend>
  <br> As dori
  <input type=text name=abon size=3
  maxlength=3 value=0 onChange="Calculeaza()">
  abonamente a 56000 lei, <br>
  adica <input type=text name=tabon size=8
              readonly value=0> lei
  incepand cu numarul <input type=text
    name=numar size=3 maxlength=3>.
  <br>
  </fieldset>
  </td> </tr>
</table>
Valoarea comenzii dumneavoastra este de
<input type=text value=0 size=9 name=total>
lei.
<table>
  <tr> <td>
  <input type=submit value='Transmite comanda'>
  </td>
  </td>
  <td>
  <input type=reset value='Anuleaza comanda'>
  </td> </tr>
</table> </form>
</body> </html>
```

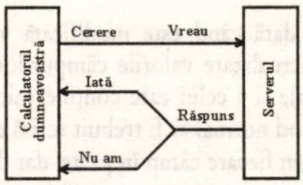
De fiecare dată când este modificată valoarea unui câmp, sunt reactualizate valorile câmpurilor care conțin totalurile parțiale și a celui care conține valoarea totală a comenzii. În mod normal ar fi trebuit scrisă câte o funcție JavaScript pentru fiecare câmp în parte, dar din motive de spațiu, am preferat să scriem o singură funcție, chiar dacă aceasta este ineficientă, în cadrul ei efectuându-se multe calcule inutile. Următoarea imagine arată că deși câmpurile care conțin totalurile sunt protejate la scriere (au atributul **READONLY**), valorile lor nu mai sunt 0 (valoarea prestabilită), ci au fost modificate pe măsură ce au fost schimbate valorile celorlalte câmpuri.



Interfața CGI

Până acum am prezentat modalitățile prin care pot fi cerute și transmise spre server anumite date. În următoarele secțiuni vă vom prezenta modul în care sunt preluate informațiile transmise cu ajutorul acestor formulare. Prelucrarea acestor date reprezintă o muncă de programare mult mai dificilă față de simpla creare a formularelor. Calea de comunicație dintre browser și serverul care găzduiește pagina Web poartă denumirea de *interfață CGI* (*Common Gateway Interface*). Programele care prelucrează informațiile din formulare pot fi scrise în orice limbaj de programare, ele având denumirea generică de *programe CGI* sau *script-uri CGI*. Majoritatea exemplelor care vor urma vor fi script-uri Unix de tip shell, deoarece este foarte probabil ca serverul care va găzdui paginile dumneavoastră să dispună de sistemul de operare Unix.

Transmiterea anumitor informații, cum ar fi afișarea datei și orei curente, cursul valutar al Băncii Naționale, ultimele știri etc. nu pot fi realizate la nivelul documentului HTML, ci numai la nivelul serverului care găzduiește acest document. Toate browserele schimbă informații cu serverele Web prin intermediul unui protocol numit *HTTP* (*Hyper-Text Transfer Protocol*). Acest protocol definește interacțiunea dintre browser și server, interacțiune care poate fi exprimată prin cereri de tip *Vreau* emise de browser și răspunsuri de tip *Iată* sau *Nu am* emise de server. Acest protocol seamănă cu ceea ce ilustrează figura următoare:



Calculatorul pe care se află browserul solicită serverului un anumit fișier (document HTML, imagine, fișier care conține codul unui applet Java sau al unui control ActiveX sau orice alt tip de fișier), iar serverul răspunde fie prin transmiterea fișierului dorit, fie printr-un mesaj de eroare care indică imposibilitatea serverului de a satisface cererea. Să presupunem că doriți să încărcați o pagină Web a cărei sursă se află în fișierul `test.html` și care conține două imagini **`IMG1.GIF`**, **`IMG2.GIF`** și un applet Java **`APPLET.CLASS`**. Un posibil scenariu al dialogului dintre browser și server este următorul:

Browser:	*Vreau fișierul* `test.html`!
Server:	*Iată fișierul* `test.html`!
Browser:	*Vreau fișierul* `img1.gif`!
Server:	*Iată fișierul* `img1.gif`!
Browser:	*Vreau fișierul* `img2.jpg`!
Server:	*Iată fișierul* `img2.jpg`!
Browser:	*Vreau fișierul* `applet.class`!
Server:	*Nu am fișierul* `applet.class`!
	Error 440: file not found

Un ultim element esențial înaintea prezentării unor script-uri CGI este faptul că ieșirea standard a acestor programe este considerată intrare standard pentru browserul apelant, adică se consideră că ieșirea script-ului reprezintă un document HTML care va fi afișat în fereastra browserului.

Primul script CGI

Vă prezentăm acum un script CGI care, dacă este apelat, determină apariția mesajului `Hello World!` în fereastra browserului apelant.

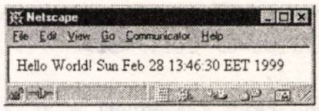
```
echo "<HTML>"
echo "<HEAD>
      <TITLE> Hello World! </TITLE> </HEAD>"
echo "<BODY> Hello World! </BODY> </HTML>"
```

Dacă vom denumi acest script `Hello.cgi` atunci apăsarea butonului *Ok* din următorul exemplu HTML va duce la afișarea mesajul `Hello World!`

```
<html> <head> <title> Ok </title>
</head> <body>
<center>
<form action=
  "http://www.ginfo.ro/skortzy/hello.cgi"
  method=get>
  <input type=submit value="Ok">
</form>
</center>
</body>
</html>
```

Alte script-uri CGI

Dacă veți dori ca în fereastra browserului să apară data și ora la care serverul a recepționat cererile browserului puteți include în fișierul `Hello.cgi` comanda Unix **`date`** care va afișa la dispozitivul standard de ieșire (intrare pentru browser) data și ora curentă. Fereastra browserului ar putea arăta astfel:



Orice schimb de informații folosind HTTP include și un set de caracteristici de mediu, accesibile din script-ul CGI. Pentru a afișa în fereastra browserului aceste caracteristici trebuie folosită comanda Unix **`printenv`**. Un script CGI care realizează acest lucru este următorul:

```
echo "<html> <head>
       <title> Mediu </title> </head>"
echo "<body> <center> <h1> Caracteristicile de
      mediu: </h1> </center>"
printenv
echo "</body> </html>"
```

Execuția acestui script duce la afișarea unor informații interesante cum ar fi *numele calculatorului* pe care se află browserul, *numele browserului.*

Cu ajutorul acestor informații puteți obține efecte interesante, de exemplu generarea de documente HTML în funcție de tipul calculatorului pe care îl utilizează vizitatorul. Să presupunem că dacă acesta utilizează un PC vrem să afișăm mesajul: `Ne pare rau, dar aceasta pagina a fost proiectata numai pentru Macintosh!`, iar dacă acesta utilizează un calculator Macintosh să afișăm: `Ne pare rau, dar aceasta pagina a fost proiectata numai pentru Windows!`. Pentru a realiza acest lucru vom căuta secvența `Mac` în șirul de caractere corespunzător numelui browserului și, în funcție de rezultatul căutării, vom afișa unul dintre cele două mesaje.

```
echo "<html> <head> <title> Dirty Trick
      </title> </head>"
x = "~echo $HTTP_USER_AGENT | grep Mac~"
echo "<body> <center>

if ["$x"=""]; then echo Ne pare rau, dar
aceasta pagina a fost proiectata numai
pentru Macintosh!
```

```
else
  echo Ne pare rau, dar aceasta pagina a fost
       proiectata numai pentru Windows!
fi
echo
</center> </body> </html>
```

Ideea de bază a acestui script este aceea de a cere sistemului Unix să caute secvența de caractere `Mac` în valoarea atribuită numelui browserului; dacă această secvență este identificată atunci variabila `x` va conține valoarea `Mac`, iar în caz contrar va conține un șir vid. Instrucțiunea **`if`** testează valoarea acestei variabile și afișează mesajul corespunzător.

Transmiterea și citirea datelor

Browserul Web va transmite serverului informațiile introduse în formular în perechi `name=value` unde `name` este valoarea atributului **`NAME`**, iar `value` conține informația corespunzătoare câmpului respectiv. Una dintre cele mai importante variabile de mediu este `QUERY_STRING` deoarece, în cazul în care metoda de transmitere a informațiilor este *get*, perechile `name=value` vor fi disponibile script-ului CGI ca valori ale acestei variabile.

Probabil că majoritatea celor care citesc aceste rânduri au folosit, cel puțin o dată, un motor de căutare pentru a găsi informații în Internet. Să presupunem că ați vizitat locația Yahoo și doriți să căutați cuvântul *ginfo*. Veți introduce acest cuvânt în caseta destinată acestui scop, apoi veți efectua un click pe butonul *Search*. După câteva secunde va apărea o nouă pagină Web care va conține rezultatul căutării. Probabil că puțini dintre dumneavoastră s-au uitat la URL-ul noii pagini Web care arată puțin ciudat:

http://search.yahoo.com/bin/search?p=ginfo.

Caracterul `'?'` indică faptul că se va transmite serverului o anumită informație suplimentară, iar `p=ginfo` este informația transmisă. Pentru programul CGI de pe serverul specificat în atributul **`ACTION`** al tag-ului **`<FORM>`** variabila `QUERY_STRING` va conține exact informația specificată după semnul de întrebare.

Datorită faptului că un URL nu poate conține anumite caractere, acestea trebuie codificate dacă se dorește transmiterea lor în acest mod. În tabelul 1 sunt prezentate codificările pentru câteva caractere uzuale.

Datorită acestor codificări, prima sarcină a script-urilor CGI este aceea de a decodifica informațiile trimise. În exemplul următor este folosit un program numit ***sed***, căruia i se cere să înlocuiască în variabila `QUERY_STRING` toate secvențele `%3A` cu două puncte și toate secvențele `%2F` cu un *slash*.

Tabel 1

Car.	Cod.	Car.	Cod.	Car.	Cod.	Car.	Cod.
' '	+	$	%24	(	%28	:	%3A
!	%21	%	%25	)	%29	=	%3D
"	%22	&	%26	+	%2B	?	%3F
#	%23	'	%27	/	%2F	\	%5C

```
if ["$QUERY_STRING"=""];
then echo "Nu exista informatii suplimentare"
else
  url="'echo $QUERY_STRING | sed \
  's/%3A/:/g;s/%2F/\//g; s/URL=//;s/url//'~"
fi
```

Un alt artificiu pe care îl permit script-urile CGI este acela de a transfera vizitatorul care apelează script-ul, la o anumită adresă. Acest lucru poate fi realizat cu ajutorul unei comenzi asemănătoare cu:

```
echo "Location: http://www.ginfo.ro/"
```

Preluarea informațiilor din formulare

Programele CGI care prelucrează formulare sunt destul de greu de realizat, deoarece depind foarte mult de sistemul de operare utilizat pe server. De exemplu, un script care realizează o anumită acțiune diferă substanțial dacă este scris pentru Unix, Windows NT, Macintosh, Amiga etc. Cea mai mare diferență o constituie limbajul de programare utilizat. Pentru Unix puteți crea un script Unix de tip shell, un script Pearl sau un program C, în timp ce pentru Macintosh va trebui să utilizați AppleScript sau MacPearl. Se recomandă proiectarea acestor script-uri folosind un limbaj portabil cum ar fi Pearl sau, mai nou, JAVA.

În cazul formularului prin care puteți comanda GInfo on-line, variabila `QUERY_STRING` va avea următoarea valoare:

```
nume=Scortaru&prenume=Mihai-Emilian&adresa=
Str.+Teleorman+Nr.+65+Ap.+46+Cluj-Napoca&
email=scort%40licj.soroscj.ro&telefon=092%2F3
27635&v8n1=1&tv8n1=5000&v8n2=2&tv8n2=10000&v8
n3=3&tv8n3=15000&v8n4=4&tv8n4=32000&v8n6=5&tv
8n6=40000&v8n7=6&tv8n7=48000&v8n8=7&tv8n8=700
00&v9n1=8&tv9n1=80000&v9n2=9&tv9n2=90000&abon
=10&tabon=560000&numar=9%2F3&total=950000
```

Pentru a prelucra aceste date trebuie apelat un program căruia să-i fie transmis ca parametru valoarea variabilei `QUERY_STRING`. Cel mai simplu astfel de program, ar stoca informațiile pe disc într-un fișier text. Iată codul sursă al unui astfel de program, scris în limbajul C:

```
#include <stdio.h>
void main(int narg, char** args){
  FILE *f=fopen("comenzi.txt","at");
  fprintf(f,"%s\n",args[1]);
  fclose(f);
}
```

Pentru mai multe informații despre script-urile CGI puteți vizita una din următoarele pagini Web:

- http://www.yahoo.com/Computers/World_Wide_Web/ Programming
- http://www.intuitive.com/CGI/

Mihai Scorțaru este student la Universitatea Tehnică din Cluj; poate fi contactat prin e-mail la adresa: scort@licj.soroscj.ro

GRAFICĂ în 256 de CULORI

Horea Haitonic

Pentru a realiza o aplicație având conținut grafic, în cazul limbajului Pascal este necesară o bibliotecă puternică de funcții și proceduri grafice. (Standardul BGI este complet nesatisfăcător, având doar 16 culori și o viteză de lucru extrem de redusă.) Unit-ul pe care îl vom prezenta implementează grafica în modul video 320×200 având 256 culori, este foarte rapid, ușor de programat și compatibil cu marea majoritate a plăcilor grafice actuale.

Biblioteca realizată în unit-ul care se prezintă în acest număr nu folosește întreruperile (mult prea lente), ci lucrează direct cu memoria ecran (`$0:$A000`). Ea poate fi folosită pentru proiectarea unor jocuri și demo-uri grafice. Interfața este identică celei de la Borland.

Primele proceduri sunt evident, cele pentru inițializarea (`SetVGA`) și respectiv, închiderea (`SetText`) modului grafic. Acestea sunt realizate folosind întreruperea `10h` (servicii video) și punând în `ax 13h` (modul grafic VGA 256) și respectiv `03h` pentru revenirea în mod text.

Inițializarea, respectiv închiderea modului grafic

```
Procedure SetVGA;
begin
  asm
    mov ax,0013h
    int 10h { initializez modul grafic 13h }
  end;
  Screen:=VGAScreen;
  Color:=0;                { culoarea curenta }
  FillColor:=0;          { culoarea de umplere }
  BkColor:=0;              { culoarea fondului }
  cpx:=0; cpy:=0;            { coordonatele CP }
end;

Procedure SetText;
begin
  asm
    mov ax, 0003h
    int 10h    { initializez modul text 03h }
  end
end;
```

Subprogramele pentru selectarea sau determinarea culorii curente, a culorii de umplere și a culorii fondului sunt extrem de simple.

Pentru lucrul cu paleta grafică, folosim porturile, accesabile în Turbo Pascal prin tablourile `port` și `portw`. Pentru obținerea componentelor roșu (`r`), verde (`g`) și albastru (`b`), punem în portul `$3c7` culoarea respectivă și în `$3c9` obținem exact componentele căutate:

Pentru setarea componentelor `r, g, b` punem în portul `$3c8` culoarea, iar în `$3c9`, pe rând, noile valori pentru roșu, verde și albastru:

Proceduri pentru lucrul cu culorile

```
Procedure GetColorPalette
                (col:Byte; var r,g,b:Byte);
begin
  port[$3c7]:=col;
      { folosesc porturile pentru obtinerea }
      { componentelor r, ,g, b              }
  r:=port[$3c9]; g:=port[$3c9]; b:=port[$3c9]
end;

Procedure SetColorPalette
                (col:Byte; var r,g,b:Byte);
begin
  port[$3c8]:=col;
       { folosesc porturile pentru setarea }
       { componentelor r, ,g, b            }
  port[$3c9]:=r;
  port[$3c9]:=g;
  port[$3c9]:=b
end;
```

Urmează procedurile ce implementează ecranul virtual, adică o zonă de memorie folosită ca un al doilea ecran grafic, foarte util pentru realizarea unei animații fluente, după cum urmează:

1. se inițializează ecranul virtual cu `SetupVirtual`;
2. se desenează în ecranul VGA principal prima poziție a animației;
3. se stabilește ecranul virtual ca ecran activ, folosind:
   `SetActiveScreen(VirtualScreen).`
   De acum desenarea nu se mai face pe monitor, ci în ecranul virtual;
4. se desenează în ecranul virtual următoarea poziție;
5. se afișează ecranul virtual pe monitor cu `ShowVirtual`;
6. se face o pauză pentru ca ochiul să perceapă imaginea; (`Delay(30)`)
7. se revine la pasul 3;
8. la terminarea animației, restabilim ecranul principal ca ecran activ (VGA), folosind:
   `SetActiveScreen(VirtualScreen).`
   Dacă nu mai avem nevoie în continuare de ecranul virtual, îl putem elibera (`FreeVirtual`).

Folosind această tehnică, se elimină efectul de clipire (*flicker*) care apare la o animație cu un singur ecran.

Implementarea procedurilor care lucrează cu ecranul virtual este ceva mai dificilă. Pentru început alocăm 64Kb de memorie pentru variabila pointer `VirtScr`, în procedura `SetupVirtual`:

Proceduri de lucru cu ecranele virtuale

```
Procedure SetupVirtual;
begin
  GetMem(VirtScr,64000);  { se alocă 64 Kb }
       { de memorie pentru ecranul virtual }
  VirtualScreen:=Seg(VirtScr^)
end;

Procedure FreeVirtual;
begin
  FreeMem(VirtScr,64000) { eliberez memoria }
          { alocata pentru ecranul virtual }
end;

Procedure SetActiveScreen(sc:Word);
begin
  if (sc<>VGAScreen) and (sc<>VirtualScreen)
  then Exit;
  Screen:=sc      { ecranul activ devine sc }
end;

Procedure ShowVirtual;
begin
  Move(VirtScr^, mem[VGAScreen:0],64000)
                    { afisare ecran virtual }
end;
```

În `VirtualScreen` am memorat adresa de segment a ecranului virtual.

Procedura `SetActiveScreen` se apelează având ca parametru tocmai adresa de segment a ecranului care va deveni activ: fie constanta `VGAScreen` (egală cu `$A000`), fie variabila `VirtualScreen` (inițializată în `SetupVirtual`).

Afișarea ecranului virtual presupune simpla copiere a 64Kb din `VirtScr^` la adresa `VGAScreen:0`.

În sfârșit, eliberarea ecranului virtual se face eliberând memoria folosită de `VirtScr` în procedura `FreeVirtual`.

Funcția de curățare a ecranului activ este foarte simplă: umplem zona de memorie alocată ecranului cu culoarea fondului (`ClearDevice`).

Pentru desenarea unui punct (pixel) pe ecran, există procedura `PutPixel`. Aceasta calculează adresa în memoria ecran a octetului corespunzător pixelului (OffSetul este `x+320*y`) și atribuie acestui octet culoarea punctului:

Proceduri pentru lucrul cu ecranul

```
Procedure ClearDevice;
begin
  FillChar(mem[Screen:0],64000,BkColor)
 { umplu memoria video cu culoarea fondului }
end;

Procedure PutPixel(x,y:Integer;c:Byte);
begin
  if (x<0) or (x>319) or (y<0) or (y>199)
  then Exit;     { punct in afara ecranului }
  mem[Screen:x+(y*320)]:=c
        { x+(y*320) este adresa in memoria }
        { video a pixelului x, y           }
end;

Function GetPixel(x,y:Integer):Byte;
begin GetPixel:=mem[screen:x+(y*320)] end;

Function GetX:Integer;
begin GetX:=cpx end;

Function GetY:Integer;
begin GetY:=cpy end;

Procedure MoveTo(x,y:Integer);
begin cpx:=x; cpy:=y end;

Procedure SetColor(c:Byte);
begin color:=c end;

Procedure SetBkColor(c:Byte);
begin BkColor:=c end;

Procedure SetFillColor(c:Byte);
begin FillColor:=c end;
```

Pentru desenarea unei linii se folosește o variantă a algoritmului *Bresenham*. Aceasta stochează în x1 și y1 coordonatele curente la care s-a ajuns în trasare, efectuând pentru fiecare pixel două adunări (se adună de fiecare dată devierile precalculate la începutul procedurii).

Algoritmul de trasare a unui arc de cerc (procedura Arc) este foarte simplu: punem pixelii aflați la distanța r de centru. Razele determină unghiul u cu orizontala (u variază între unghiul inițial și unghiul final):

Pentru desenarea unui sector de cerc, desenăm mai multe arce concentrice cu raza rr din ce în ce mai mare.

Desenarea unui dreptunghi este de asemenea foarte simplă; trasăm linii între cele patru vârfuri. Pentru un dreptunghi haşurat, desenăm un dreptunghi gol și apoi linii orizontale având culoarea de umplere.

Primitive grafice

```
Procedure Line(x1,y1,x2,y2:Integer);
  { acesta este un algoritm complex si      }
  {destul de rapid de trasare a unei linii  }
var u,s,v,d1x,d1y,d2x,d2y,m,n:Real;
    i:Integer;
begin
  u:=x2-x1; v:=y2-y1;
  d1x:=Sgn(u); d1y:=Sgn(v);
  d2x:=Sgn(u); d2y:=0;
  m:=Abs(u); n:=Abs(v);
  if not (m>n)
  then
  begin
    d2x:=0; d2y:=sgn(v); m:=Abs(v); n:=Abs(u)
  end;
  s:=int(m/2);
  for i:=0 to Round(m) do
  begin
    PutPixel(x1,y1,color);
    s:=s+n;
    if not (s<m)
    then
    begin
      s:=s-m;
      x1:=x1+Round(d1x); y1:=y1+Round(d1y)
    end
    else
    begin
      x1:=x1+Round(d2x); y1:=y1+Round(d2y)
    end
  end
end;

Procedure Arc(x,y,ui,uf,r:Integer);
var xx,yy,u,du:Integer;
begin
  u:=ui; du:=Round(15/r);
              { pasul cu care marim unghiul }
  if du=0 then du:=1;
  repeat
    xx:=Round(r*cos(2*u));
      { coordonata x a punctului urmator }
    yy:=Round(r*sin(2*u));
      { coordonata y a punctului urmator }
    PutPixel(x+xx,y+yy,Color);
    Inc(u,du);
  until u>uf
end;

procedure Pieslice(x,y,ui,uf,r:Integer);
    { sector de cerc cu centrul x, y intre }
    { unghiurile ui-uf si cu raza r        }
var xx,yy,u,du,rr:Integer;
    OldColor:Byte;

begin
  Arc(x,y,ui,uf,r);
              { arcul cu raza cea mai mare }
  OldColor:=Color; { salvam culoarea curenta }
  SetColor(FillColor);
          { trasam cu culoarea de umplere! }
  for rr:=1 to r-1 do Arc(x,y,ui,uf,rr);
  SetColor(OldColor) { refacem culoarea curenta }
end;

procedure Rectangle(x1,y1,x2,y2:Integer);
begin
  Line(x1,y1,x2,y1); Line(x2,y1,x2,y2);
          { unim cele patru varfuri prin  }
          { linii orizontale si verticale }
  Line(x2,y2,x1,y2); Line(x1,y2,x1,y1)
end;

procedure Bar(x1,y1,x2,y2:Integer);
var temp:Integer;
    OldColor:Byte;
begin
  if (x1>x2)
  then begin temp:=x1; x1:=x2; x2:=temp end;
  { inversam coordonatele pentru a avea in  }
  { x1, y1 coordonatele coltului stanga sus }
  if (y1>y2)
  then begin temp:=y1; y1:=y2; y2:=temp end;
  Rectangle(x1,y1,x2,y2);
  OldColor:=Color;
  SetColor(FillColor);
  Inc(y1);
  if x1=x2 then Exit;
  repeat
    Line(x1+1,y1,x2-1,y1);
    { trasam linii orizontale de sus in jos }
    Inc(y1)
  until y1>=y2;
  SetColor(OldColor)
end;
```

Urmează funcțiile pentru lucrul cu imagini. Prima dintre ele este funcția care întoarce dimensiunea imaginii (în octeți) (`ImageSize`). Pentru aceasta, aflăm lungimea `l` și înălțimea `h` a imaginii.

```
l:=Abs(x2-x1+1); h:=Abs(y2-y1+1);
```

Dimensiunea este numărul pixelilor imaginii la care se adaugă 4 (câte 2 octeți pentru lățimea și lungimea imagnii).

```
ImageSize:=l*h+4;
```

Procedura `GetImage` salvează imaginea conținută în dreptunghiul `(x1,y1,x2,y2)` în memorie, în variabila `Buffer` (de obicei un pointer pentru care s-au alocat `ImageSize(x1,y1,x2,y2)` octeți).

Pentru afișarea pe ecran a unei imagini salvate cu `GetImage` folosim procedura `PutImage`.

Proceduri pentru lucrul cu porțiuni de imagini

```
Function ImageSize(x1,y1,x2,y2:Integer):Word;
var l,h:Integer;
begin
  l:=Abs(x2-x1+1);
  h:=Abs(y2-y1+1);        { dimensiunea este }
       { numarul de pixeli ai imaginii +4 }
  ImageSize:=l*h+4
end;

Procedure GetImage(x1,y1,x2,y2:Integer;
                                  var Buffer);
var segm,offs,s:Word;
    x,y,l,h:Integer;
begin
  l:=Abs(x2-x1+1); h:=Abs(y2-y1+1);
  segm:=seg(Buffer);
  offs:=Ofs(Buffer);
  mem[segm:offs]:=l and 255;
  mem[segm:offs+1]:=l shr 8;
       { copiez fiecare linie in zona de  }
       { memorie alocata variabilei bitmap }
  mem[segm:offs+2]:=h;
  for y:=0 to h-1 do
    Move(mem[Screen:x1+320*(y+y1)],
                    mem[segm:l*y+4+offs],l+1)
end;

Procedure PutImage(x,y:Integer; var Buffer);
var segm, offs:Word;
    l,h,yy:Integer;
begin
  Segm:=seg(Buffer); Offs:=Ofs(Buffer);
  l:=mem[segm:offs+1]shl 8 or mem[segm:offs];
  h:=mem[segm:offs+2];{ l si h sunt integer }
            { si se memoreaza pe 2 octeti  }
  for yy:=0 to h-1 do
    Move(mem[segm:l*yy+4+offs],
                 mem[Screen:320*(yy+y)+x],l)
end;
```

Formatul de imagine ales (extrem de simplu) este identic celui prevăzut de BGI: 4 octeți pentru dimensiunile orizontale și verticale ale imaginii și în continuare pixelii, unul după altul, parcurgând imaginea pe orizontală.

Partea de interfață a unit-ului

```
unit HVGA;

interface

type rgb=record
          r,g,b:Byte;
        end;
     palette=array[0..255] of rgb;
const VGAScreen:Word=$A000;
var VirtualScreen:Word;
    GraphErrorMsg:String;

function  Sgn(a:Real):Integer;
procedure SetVGA;
         { initializeaza modul grafic VGA256}
procedure SetText;   { inchidere mod grafic }
function  GetX:Integer; { coordonata x a CP }
function  GetY:Integer; { coordonata y a CP }
procedure MoveTo(x,y:Integer);
                  { muta CP la pozitia x, y }
procedure SetColor(c:Byte);
              { seteaza culoarea de scriere }
procedure SetBkColor(c:Byte);
                { seteaza culoarea fondului }
procedure SetFillColor(c:Byte);
              { seteaza culoarea de umplere }
procedure GetColorPalette(col:Byte;
     var r,g,b:Byte); { obtine componentele }
                { r, g, b ale culorii col }
procedure SetColorPalette(col:Byte;
    var r,g,b:Byte); { seteaza componentele }
                { r, g, b ale culorii col }
procedure SetupVirtual;
               { initializare ecran virtual }
procedure FreeVirtual;
    { elibereaza memoria ecranului virtual }
procedure SetActiveScreen(sc:Word);
                  { stabilesc ecranul activ }
procedure ShowVirtual;
                 { afisare ecran virtual }
procedure ClearDevice;{ sterge ecranul activ }
procedure PutPixel(x,y:Integer; c:Byte);
              { pixel de culoarea c la x, y }
function  GetPixel(x,y:Integer):Byte
          { obtine culoarea pixelului x, y }
procedure Line(x1,y1,x2,y2:Integer);
            { linie de la x1, y1 la x2, y2 }
procedure Arc(x,y,ui,uf,r:Integer);
       { arc de cerc cu centrul x, y intre }
       { unghiurile ui-uf si cu raza r     }
```

```
procedure Pieslice(x,y,ui,uf,r:Integer);
      { sector de cerc cu centrul x, y intre }
      { unghiurile ui-uf si cu raza r        }
procedure Rectangle(x1,y1,x2,y2:Integer);
                                  { dreptunghi }
procedure Bar(x1,y1,x2,y2:Integer);
                          { dreptunghi umplut }
function  ImageSize(x1,y1,x2,y2:Integer):Word;
       { dimensiunea (in bytes) a imaginii }
procedure GetImage(x1,y1,x2,y2:Integer;
                                  var Buffer);
             { copiaza in memorie imaginea }
procedure PutImage(x,y:Integer; var Buffer);
          { afiseaza imaginea din memorie }

implementation

uses crt;
type VirtualScreenType=array[1..64000] of Byte;
     VirtPtr=^VirtualScreenType;

var Screen:Word;
    Color,BkColor,FillColor,TextSize:Byte;
    cpx,cpy:Integer;
    VirtScr:VirtPtr;

function Sgn(a:Real):Integer;
begin
  if a<0
  then sgn:=-1
  else
    if a=0
    then sgn:=0
    else sgn:=1
end;

          { urmeaza procedurile prezentate }
          { pe parcursul articolului       }

End.
```

Unitul poate fi îmbunătățit, adăugând spre exemplu proceduri și funcții specializate pentru desenarea diferitelor forme geometrice, proceduri de salvare a imaginilor și paletelor pe disc, proceduri care realizează efecte ca *fade-in*, *fade-out*, suprapunere de imagini mascate etc.; aceste noi facilități sunt relativ ușor de implementat, având procedurile de bază.

Horea Haitonic este elev la Liceul "George Coșbuc" din Năsăud; poate fi contactat prin intermediul redacției.

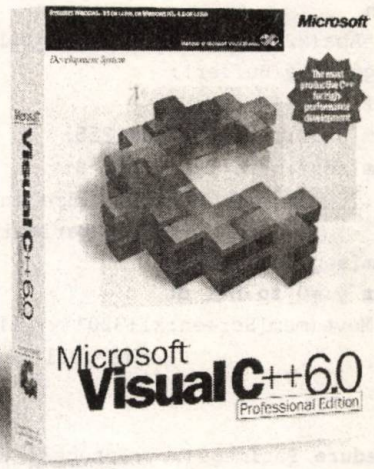
PREMIUL LUNII

Și în acest număr, revista noastră oferă un *PREMIU DE FIDELITATE* unui cititor ales prin tragere la sorți dintre *ABONAȚI*.

Și tu ai șansa să câștigi *PREMIUL LUNII*. Condiția este simplă: *ABONEAZĂ-TE !*

Premiul lunii martie este oferit de sponsorul nostru, *MICROSOFT ROMANIA*...

Descoperiți modul de lucru Visual



... și a fost câștigat de către
Benchea Nicoleta
din Aiud

Microsoft®

Din secretele ȘIRURILOR

Sorin Groza

Analiza problemelor de informatică este o etapă care nu trebuie "sărită" niciodată. Pentru a vă convinge, vă prezentăm câteva "artificii" care îmbunătățesc în mod considerabil timpul de execuție al algoritmilor. Nu vă speriați, știm că această parte a GInfo se adresează "piticilor"; nu vom intra în prea multe detalii.

Problema 1: Numărul lipsă

Se citește de la tastatură un număr natural `n` *și apoi* `n` *numere naturale distincte din intervalul* `[0,n]`. *Deoarece în intervalul* `[0,n]` *sunt* `n+1` *numere naturale, rezultă că unul dintre numere lipsește. Scrieți un program care afișează numărul absent.*

Date de intrare:

```
n=7
2 6 1 4 7 0 5
```

Date de ieșire:

```
lipseste numarul 3
```

Rezolvare:

Prima idee de rezolvare a problemei constă în a păstra într-un șir toate cele `n` numere și apoi, pentru fiecare număr de la 0 la `n`, trebuie verificat dacă este prezent sau nu în șir. În momentul în care se descoperă primul număr absent, acesta se va afișa, deoarece se știe că lipsește unul singur.

O altă idee este de a ordona crescător șirul numerelor. Se verifică apoi diferența dintre fiecare două elemente consecutive. Când aceasta este diferită de `1`, atunci a fost găsit numărul lipsă.

Ambii algoritmi au un timp de răspuns mare, pentru o valoare mare a lui `n`.

Vom prezenta o implementare bazată pe prima idee, dar care *nu necesită căutarea fiecăruia dintre numere*, ceea ce ar putea însemna, în cazul cel mai defavorabil, parcurgerea integrală a șirului dat, de `n` ori. Pentru a evita aceste parcurgeri, vom folosi un șir `v` având `n` elemente, în care elementul `v[i]` indică prezența numărului `i` în șir. În momentul introducerii valorii `i` în șirul `a`, `v[i]` devine `true`. Este clar că în final toate elementele șirului vor fi egale cu `true`, mai puțin unul care rămâne cu valoarea `false`.

Pentru a afla care este numărul absent, vom parcurge șirul `v`, iar elementul `v[i]=false` semnalează faptul că numărul `i` nu există în șirul dat.

Elemente

Conform definiției, un *șir* (tablou unidimensional) este *o mulțime finită de elemente de același tip care are asociat un nume*.

Un element al șirului este identificat prin intermediul *numelui* șirului și al unui *indice*.

Pentru a folosi un șir într-un program Pascal acesta trebuie declarat în secțiunea de variabile. Iată o astfel de declarație:

```
var pret:array[1..100] of Integer;
```

Șirul cu numele `pret` și care va avea cel mult 100 de elemente poate fi utilizat, de exemplu, pentru memorarea prețului a cel mult 100 de mașini prezente la expoziția de la New York.

În general, în problemele de informatică, elementele șirurilor sunt diverse numere (naturale sau întregi) care pot reprezenta înălțimile elevilor unei clase, sume de bani, valorile unor cadouri, greutățile unor persoane etc.

Evident, există foarte multe modalități de utilizare a șirurilor. Elementele unui șir pot indica dacă are sau nu loc un eveniment (situație în care vor avea valori doar de 0 sau 1) sau numărul de apariții ale unui eveniment. În cazul în care doar prezența/absența ne interesează, sunt foarte utile șirurile având valori logice (booleene).

Listing: ABSENT.PAS

```
Program Absent;
const n_max=100;
var n,i:Integer;
    gasit:Boolean;
    v:array[0..n_max] of Boolean;
    a:array[0..n_max] of Integer;

Begin
  Write('n= '); Readln(n);
  Writeln(n,' numere distincte');
  for i:=0 to n_max do v[i]:=false;
  for i:=1 to n do
  begin
    Read(a[i]); v[a[i]]:=true
  end;
  Readln;
  gasit:=false; i:=0;
  while not gasit do
    if v[i] then Inc(i)
            else
            begin
             Writeln('Lipseste numarul ',i);
              gasit:=true
            end;
  Readln
End.
```

Problema 2: Aeroportul

Un grup de teroriști ce pregătesc o lovitură de stat, își propun să ocupe un aeroport strategic. Ei dețin harta zborurilor între cele `n` *aeroporturi, adică* `m` *perechi* `(i,j)` *ce indică faptul că există zbor din aeroportul* `i` *în aeroportul* `j`*. Ei vor ocupa acel aeroport din care se poate ajunge în fiecare din celelalte aeroporturi, dar în care nu se poate ajunge. Scrieți un program care găsește aeroportul care trebuie ocupat.*

Date de intrare:

```
n=3
m=3
1 3
2 1
2 3
```

Date de ieșire:

```
Aeroportul ocupat este 2
```

Rezolvare:

Trebuie găsit acel aeroport `i`, corespunzător numărului `i` care apare exact de `n-1` ori în șirul care conține plecările din aeroporturi și de `0` ori în șirul care conține sosirile. Se va testa fiecare număr de la `1` la `n` dacă verifică această condiție. Când se găsește numărul, acesta se va afișa, deoarece reprezintă aeroportul căutat. Și în cazul acestei probleme se va folosi ideea de mai sus. Astfel, vom folosi 2 șiruri: un șir `plec` (`plec[i]` va arăta spre câte aeroporturi se poate zbura din aeroportul `i`); un alt șir `sos` (`sos[j]` va arăta din câte aeroporturi se poate ajunge în aeroportul `j`).

Cum se construiesc cele 2 șiruri?

Se citește o pereche `(p s)`, iar `plec[p]` și `sos[s]` se măresc cu `1`, deoarece s-a introdus faptul că din aeroportul `p` se pleacă, iar în aeroportul `s` se sosește. Se parcurg cele două șiruri, iar poziția pentru care `plec[i]=n-1` și `sos[i]=0` indică faptul că aeroportul este `i`.

Listing: AEROPORT.PAS

```
Program Aeroport;
const n_max=100;
var n,m,i,p,s:Integer;
    gasit:Boolean;
    plec,sos:array[0..n_max] of Integer;
Begin
  Write('Numarul de aeroporturi: );
  Readln(n);
  Write('Numarul legaturilor: '); Readln(m);
  for i:=1 to m do
  begin
    Write('Introduceti legatura ',i,': ');
    Read(p,s);
    Inc(plec[p]); Inc(sos[s])
  end;
  Readln;
  gasit:=false; i:=0;
  while not gasit do
  begin
    if (plec[i]=n-1) and (sos[i]=0)
    then
    begin
      Writeln('Aeroportul ocupat este ',i);
      gasit:=true
    end;
    Inc(i)
  end;
  Readln
End.
```

Problema 3: Litera preferată

Un grup de `n` *copii din clasa I sunt solicitați de către doamna învățătoare să noteze pe un bilețel litera (mică) preferată (după studiul alfabetului). Scrieți un program care afișează:*
- câte litere distincte ale alfabetului au fost folosite;
- care este litera preferată a clasei și numărul copiilor care au ales-o.

Date de intrare:

```
n=10
a b a a c b d a c b
```

Date de ieșire:

```
4 litere distincte
litera preferata: a
a fost aleasa de 4 copii
```

Rezolvare:

Pentru rezolvarea problemei vom folosi un șir `litera` cu 26 de elemente, unde `litera[i]` reprezintă de câte ori a fost aleasă litera de pe poziția `i` din alfabet (de exemplu `litera[3]` arată de câte ori a fost aleasă litera `c`). Se va

parcurge șirul `litera`, iar numărul elementelor diferite de `0` va arăta câte litere distincte au fost folosite. Aflarea literei preferate se reduce astfel la aflarea maximului șirului cu frecvența literelor (fie acesta `max`) și a poziției acestuia (notată `poz`). Litera preferată este litera care se află pe poziția `poz` în alfabet și a fost aleasă de `max` copii. Pentru exemplul nostru `poz=1`, deci litera preferată este `a` și `max=4`, adică a fost aleasă de 4 copii.

Cum se construiește șirul `litera`?

În momentul citirii unei litere `lit` se va determina poziția ei în alfabet: `pozitie=Ord(lit)-Ord('a')+1`. Valoarea elementului `litera[pozitie]` se va mări cu `1`. După citirea ultimei litere va fi construit șirul cu numărul de apariții a fiecărei litere introduse.

Listing ALFABET.PAS

```
Program Alfabet;
var n,i,max,Poz,pozitie,nr_lit:Integer;
    litera:array[1..26] of Integer;
    lit,ales:Char;
Begin
  Write('Numarul de litere ');
  Readln(n);
  for i:=1 to n do
  begin
    Write('Introduceti litera ',i,' ');
    Readln(lit);
    pozitie:=Ord(lit)-Ord('a')+1;
    Inc(litera[pozitie])
  end;
  for i:=1 to 26 do
  begin
    if litera[i]<>0 then Inc(nr_lit);
    if litera[i]>max
    then begin max:=litera[i]; Poz:=i end
  end;
  ales:=Chr(Poz+Ord('a')-1);
  Writeln(nr_lit,' litere distincte');
  Writeln('Litera preferata: ',ales);
  Writeln('a fost aleasa de ',max,' copii');
End.
```

În încheiere vă propunem câteva probleme de rezolvat, pentru a oferi posibilitatea verificării modului de aplicare a ceea ce ați învățat nou din cele prezentate anterior.

Problema 4: Moșul cel uituc

Moșul cel bun a primit scrisori de la toți copiii cuminți, scrisori pe care le-a numerotat în ordinea sosirii, începând de la `1`. Plecând la drum, se gândește că din grabă a uitat unele scrisori. Scoate din sac câte o scrisoare, una câte una, notând pe o foaie numărul fiecăreia. Știind că a primit `n` scrisori, scrieți un program care afișează cele pe care le-a uitat acasă. Se dă numărul de scrisori (`n`) și numerele scrisorilor existente în sac. Enumerarea lor se va termina cu `0` (zero).

Date de intrare:

```
n=10
8 3 6 1 7 2 10 5 0
```

Date de ieșire:

```
Scrisori uitate: 4 9
```

Problema 5: Străzi și piețe

Într-un oraș există `n` intersecții. Se citesc de la intrare perechi de forma `(i j)` cu semnificația că din intersecția `i` se poate ajunge direct în intersecția `j` fără a trece prin alte intersecții.

*Numim **piață** o intersecție în care numărul străzilor care intră, plus numărul străzilor care ies din ea este cel puțin `4`. **Piața centrală** este intersecția cu cel mai mare număr de străzi care intră și care ies (este unică). Să se afle toate pieței și să se stabilească piața centrală.*

Observație:
Sfârșitul perechilor este dat de perechea `(0 0)`.

Date de intrare:

```
n=6
1 2
1 4
1 6
2 3
2 6
3 4
4 1
4 5
4 6
5 2
5 3
0 0
```

Date de ieșire:

```
3 piete: 1 2 4
piata centrala: 4
```

Problema 6: Și iar litere

Pe plaja de la Năvodari au sosit copii care își petrec vacanța la mare. Profesorul însoțitor organizează următorul joc: copiii se împart în două grupe. Copiii din fiecare grupă vor scrie pe nisip litere ale alfabetului, obținându-se astfel două șiruri de litere. Apoi, toți copiii vor studia atent cele două șiruri de litere, dorind să descopere literele alfabetului care apar de același număr de ori în cele două șiruri. Un copil mai isteț, văzând care este jocul, scrie un program care, plecând de la cele două șiruri de litere, obține literele care apar de același număr de ori în cele două șiruri. Scrieți și voi un astfel de program.

Date de intrare:

```
msbsshbl      - primul șir
skxbbsmmsk    - al doilea șir
```

Date de ieșire:

```
Litere cu acelasi numar de aparitii:
s b
```

Domnul Sorin Groza este cadru didactic la Clubul Copiilor din Aiud; poate fi contactat prin intermediul redacției.

Vă place GInfo?

Nume: _____ Prenume: _____ Vârsta: _____

Adresa: _____

Localitate: _____ Cod poștal: _____ Județ: _____ Telefon: _____

Ocupația: elev(ă) informatică ☐ elev(ă) ☐ student(ă) ☐ cadru didactic ☐ altă ocupație ☐ (precizați:_____)

Școala (firma, organizația etc.): _____

Dotări

Folosiți un calculator acasă? Da ☐ Nu ☐

Câte calculatoare sunt instalate în școală: (aprox.) _____

Aveți acces la e-mail: Da ☐ Nu ☐

Aveți acces la WWW: Da ☐ Nu ☐

Cum apreciați dotarea cu tehnică de calcul a școlii: _____

Aprecieri generale

Acordați note de la 1 la 10 pe următoarele criterii:

Structura tematică: ___

Calitatea articolelor: ___

Prezentare grafică: ___

Calitate tipografică: ___

Considerați că prețul este:

prea mare ☐ accesibil ☐ prea mic ☐

De cât timp cunoașteți Gazeta de Informatică (ani)? _____

Cum ați procurat acest număr:

chioșc ☐ abonament ☐ cadou ☐ altfel ☐

În acest număr

Apreciați cu note de la 1 la 10 articolele acestui număr:

DirectX ___

Bacalaureat 1999 ___

Zero-polinoame ___

Criptarea cu chei publice ___

Pagini Web cu HTML 4.0 ___

Grafică în 256 de culori ___

Din secretele șirurilor ___

Articolul care v-a plăcut cel mai mult: _____

De ce? _____

Articolul care v-a plăcut cel mai puțin: _____

De ce? _____

Considerați că ponderea problemelor rezolvate este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că ponderea problemelor propuse este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că ponderea subiectelor teoretice este:

prea mică ☐ potrivită ☐ prea mare ☐

Considerați că nivelul de accesibilitate al articolelor este

prea coborât ☐ potrivit ☐ prea ridicat ☐

În viitor

În următoarele numere așteptați:

mai multe articole de programare? Da ☐ Nu ☐

mai multe articole teoretice? Da ☐ Nu ☐

mai multe articole despre Internet? Da ☐ Nu ☐

mai multe prezentări de programe? Da ☐ Nu ☐

prezentări de jocuri? Da ☐ Nu ☐

articole despre hardware? Da ☐ Nu ☐

V-ar plăcea să citiți:

pagini redactate de elevii unor licee? Da ☐ Nu ☐

lucrări premiate, realizate de elevi? Da ☐ Nu ☐

actualități școlare? Da ☐ Nu ☐

prezentarea unor situri Web? Da ☐ Nu ☐

tips & tricks? Da ☐ Nu ☐

Comentarii:

_____

_____

_____

_____

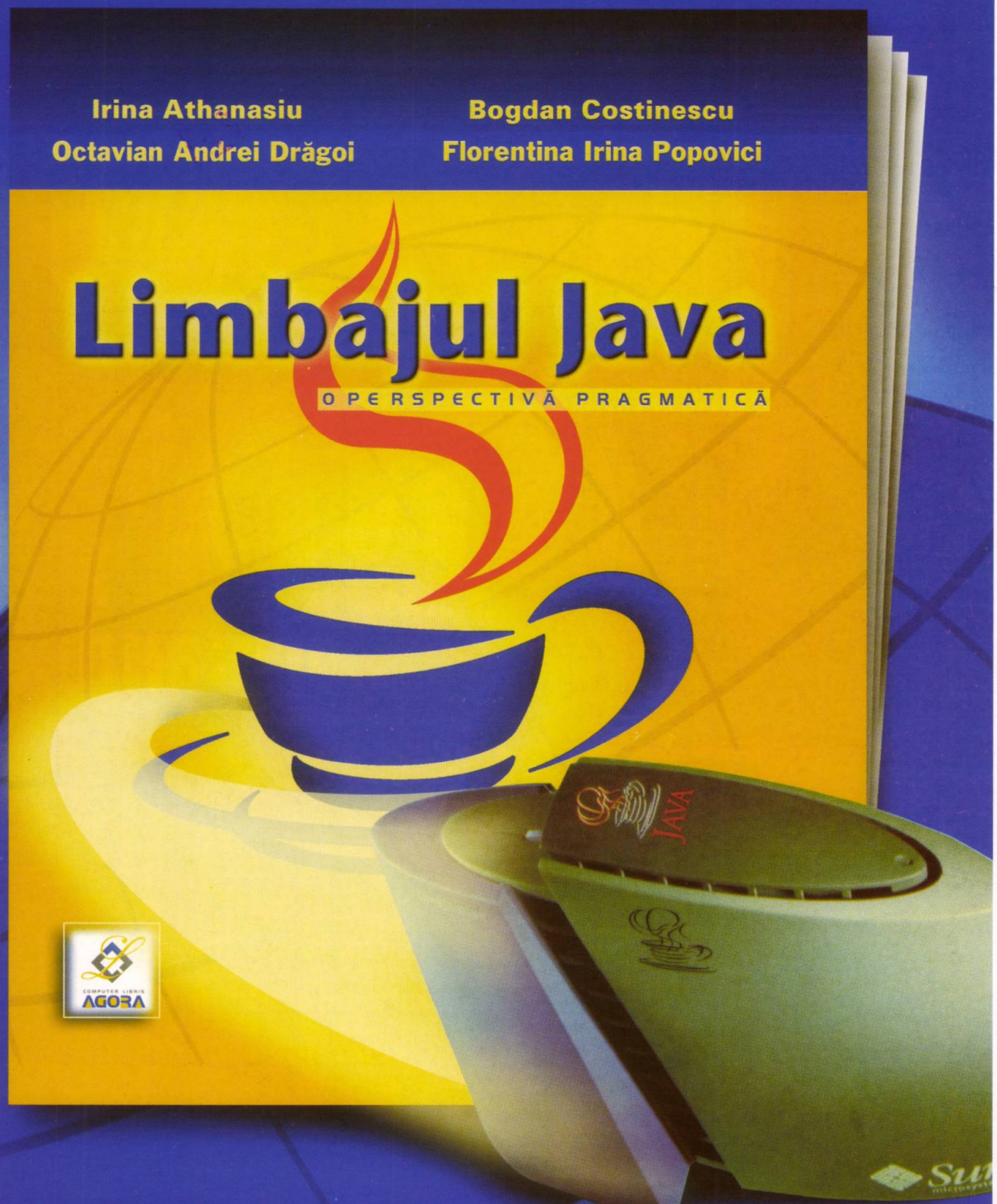
_____

_____

Dacă spațiul nu vă este suficient, folosiți coli albe suplimentare

artea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Car

COMPUTER LIBRIS AGORA



COMPUTER LIBRIS AGORA Cluj-Napoca, Str.Universității Nr.8/8, 3400, Tel. 064-192422
COMPUTER PRESS AGORA Tg.Mureș Str.Tudor Vladimirescu Nr. 63/1 (CP 172/1), Tel. 065-166516, Fax 065-166290
Sucursala București: Str.Poterași, Nr.35/2 (CP 94/49), Tel: 01-3309281, Fax: 01-3309285

Un sfat prinde bine...

PC MAGAZINE ROMÂNIA

...oricând!

Dacă vrei să te orientezi printre miile de produse și tehnologii informatice de ultimă oră

ABONEAZĂ-TE!

Câștigi **30 000** lei și vei primi cadou și CD-urile **WINSTONE '99** și **WINBENCH '99**

COMPUTER PRESS AGORA, Str.Tudor Vladimirescu Nr.63/1, 4300 Tg.Mureș, Tel.065-166516, Fax 065-166290

Mai 1999
Vol. 9/5

Lei 10.000

Gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:

REȚELE neuronale

BABEL

TEHNO

Sisteme electronice de plăți

INTERNET

Informații din CMOS

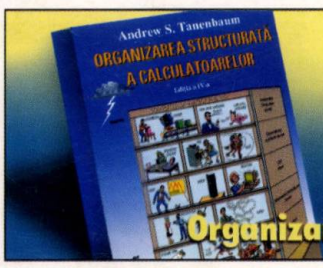
Easter Eggs

Plus

De la FORTRAN la JAVA

ONI'99

aprilie '99 Mediaș



Andrew S. Tannenbaum

PREMIUL LUNII

Organizarea structurată a calculatoarelor

COMPUTER PRESS AGORA

PC & BYTE REPORT

TEHNOLOGIA INFORMAȚIEI

ECHIPAMENTE • SISTEME • APLICAȚII • REȚELE • INTERNET

*Abonează-te și plătești doar 120.000 lei (cu 60.000 lei mai puțin decât prețul de vânzare), evitând și ulterioarele majorări de preț

COMPUTER PRESS AGORA • CP.230 OP.1 TG.MUREȘ, Tel. 065-166516 Fax 065-166290

Vol. 9, Nr. 5 1999

gazeta de informatică

Σ Sumar

probleme.. PAG. 5

propuse pentru REZOLVARE

Olimpiada Națională de Informatică - Mediaș

focus.. PAG. 18

Rețele neuronale
Lucian BRÂNCOVEAN

Despre "inteligența" calculatoarelor

tehno.. PAG. 22

Modelul relațional
Ion COZAC

Informațiile din memoria CMOS
Ion Gelu TOMȘA

internet.. PAG. 36

Criptografia în sisteme electronice de plăți
Victor-Valeriu PATRICIU

Pentru mai multă siguranță

profil.. PAG. 41

De la FORTRAN la JAVA
Gabriela STROE

Evoluția limbajelor de programare

babel.. PAG. 44

Easter Eggs
Gabriela STROE

Surprize software

serial.. PAG. 48

Pagini WEB cu HTML 4.0
Mihai SCORȚARU

Ultimul episod...

begin.. PAG. 61

Algoritmi elementari
Sorin GROZA

Să învățăm să programăm

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Au mai colaborat:
Victor-Valeriu Patriciu (București)
Ion Cozac (Tg. Mureș)
Sorin Groza (Aiud)
Gabriela Stroe (Cluj)
Ion Gelu Tomșa (Dej)
Lucian Brâncovean (Sibiu)

Tehnoredactare:
Dan Crețu, Călin Borz
Coperta și grafica:
Mircea Drăgoi

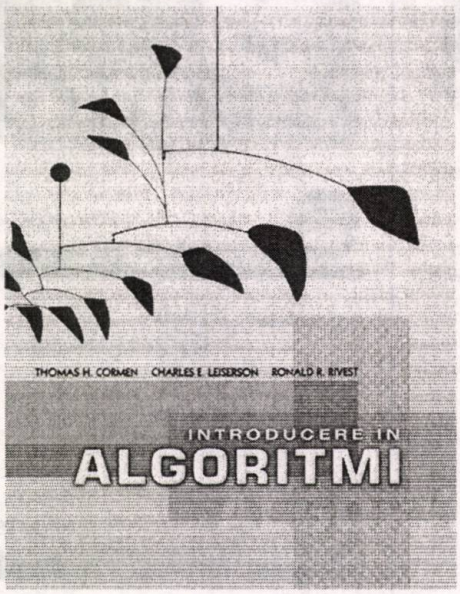
Adresa pentru corespondență:
C.P. 230-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: (065) 166.516
Fax: (065) 166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *Master Druck*
Târgu-Mureș
tel: 065/162.359
Editor: Computer Press Agora srl
str. T. Vladimirescu nr. 63/1
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director editorial:
Mircea Sârbu
Director tehnic:
Szabó László
Vânzare reclamă:
Gabriela Bucșa (gbucsa@agora.ro)
Denis Mărioara (mdenis@agora.ro)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: (01) 330.92.81

COMPUTER LIBRIS AGORA

În curând o nouă apariție editorială

La începutul lunii mai va apare în sfârșit și în limba română celebra carte de ALGORITMI scrisă de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, publicată de MIT Press, SUA. Este o carte care de la prima ei apariție a fost retipărită, numai în Statele Unite, de 21 ori.



Această carte oferă o introducere exhaustivă a studiului modern al algoritmilor. Analiza și proiectarea algoritmilor tratați este accesibilă și asigură condițiile unui studiu, pe de o parte fundamental, pe de altă parte profund.

Fiecare capitol prezintă cel puțin un algoritm sau o tehnică de proiectare, un domeniu de aplicații, respectiv un subiect înrudit. Algoritmii sunt descriși în limba română și în pseudocod. Cartea conține peste 260 de figuri care ilustrează modul de funcționare al algoritmilor. Deoarece accentul este pus pe eficiență ca și criteriu de proiectare, include analize atente ale timpului de executare a tuturor algoritmilor.

Cartea originală conține 1000 pagini și este structurată în șapte părți, subdivizate în 37 capitole, cu o bibliografie de 205 titluri și o listă bibliografică de 30 pagini.

Partea I, "Fundamente matematice", este un compendiu al metodelor și instrumentelor care vor fi utilizate în analiza algoritmilor. Este organizată esențial pentru referință, ca un fel de tutorial.

Partea a II-a, "Sortare și statistici de ordine", tratează problematica tehnicilor de sortare. Sunt studiate metodele *heapsort, quicksort și tehnici de sortare în timp liniar, precum și statisticile de ordine.*

Partea a III-a, "Structuri de date", prezintă unele tehnici de bază pentru reprezentarea și manipularea mulțimilor dinamice finite. Sunt discutate *structuri de date elementare, tabele de dispersare, arbori de căutare binară, arbori roșu-negru, extinderea structurilor de date.*

Partea a IV-a, "Tehnici avansate de proiectare și implementare", acoperă trei tehnici importante pentru proiectarea și analiza algoritmilor eficienți: *programarea dinamică, algoritmi greedy și analiza amortizată.* Aceste tehnici sunt ceva mai sofisticate decat altele, dar sunt esențiale pentru faptul că permit abordarea efectivă a multor probleme computaționale.

Partea a V-a, "Structuri de date avansate", revine la examinarea structurilor de date care suportă operații pe mulțimi dinamice, dar la un nivel mai avansat decat cel din Partea a III-a. Sunt discutați *B-arbori, ansambluri binomiale, ansambluri Fibonacci, precum și structuri de date pentru mulțimi disjuncte.*

Partea a VI-a, "Algoritmi de grafuri", tratează unele dintre cele mai importante probleme computaționale definite pe grafuri. Sunt trecuți în revistă algoritmi elementari pe grafuri, arbori de acoperire minimă, problematica celor mai scurte drumuri și algoritmi pentru determinarea fluxului maxim.

Partea a VII-a, "Subiecte diverse", conține o selecție de subiecte care extinde și completează materialul din restul cărții. Unele capitole introduc modele de calul noi precum circuite combinatoriale sau computere paralele. Altele acoperă domenii mai specializate precum geometria computațională sau teoria numerelor. Ultimele două capitole discută despre unele dintre limitările cunoscute ale proiectării algoritmilor eficienți și introduc tehnici pentru studierea acestor limitări.

COMPUTER LIBRIS
AGORA

Pentru comenzi sunați la Computer Libris AGORA [Tel: 064-192422]

Între OLIMPIADE și VACANȚA MARE

Eterna bucurie a școlarilor, dar și a profesorilor este vacanța mare. Ea se apropie... ce-i drept, cam încetișor. Mi-am adus aminte de ea, constatând că acest număr al Ginfo este ultimul pe acest an școlar. Așteptăm vacanța în fiecare an, copii și adulți, visând la frumusețile unor decoruri montane sau la plaja întinsă de pe litoral, la dimineți în care nu sună ceasul și la după-amieze în care nu trebuie făcute teme...

Pentru copiii care anul viitor își continuă studiile este o pauză (bine meritată — sper); pentru absolvenții de liceu este un sfârșit de an mai greu ca oricare altul, dar sunt și alții care au "câte ceva" de făcut în vacanța care vine.

- Absovenții au BAC în iunie-iulie, apoi în toamnă, admitere. Nu va fi un BAC ușor, știm de cel puțin doi ani. Pasionații acestei științe au posibilitatea să dovedească (și sunt convinsă că o vor face) că nu au învățat *informatică* opt ore pe săptămână *degeaba.* Pentru că acest BAC este primul din România, când se poate da examen și la informatică. Le doresc succes, amintindu-le totodată că de "realizarea" lor depinde într-o oarecare măsură și viitorul acestei discipline în strategia învățământului preuniversitar.

- Dacă toți absolvenții vor da BAC, acest lucru nu este adevărat și pentru admitere. Campionii olimpiadelor școlare, în funcție de instituția de învățământ superior aleasă, vor "intra" la facultate fără examen de admitere. Sunt mai multe variante, conform cărora, "contează" un premiu obținut la olimpiada națională, dar și unul de la olimpiada județeană. Felicităm decizia Ministerului Educației Naționale și a instituțiilor de învățământ superior, care astfel demonstrează că în sfârșit și au adus aminte și de *talentele națiunii.*

- Acest număr al *Ginfo* a "așteptat" festivitatea de închidere a *Olimpiadei Naționale de Informatică*, deoarece am dorit să publicăm problemele propuse concurenților, cu această ocazie. Nu doar de dragul publicării. Cei interesați pot răsfoi revista și în vacanță și — cine știe — într-o zi ploioasă sau "plicticoasă" pot încerca o rezolvare din curiozitate sau de dragul senzației de succes. "Enunțul problemelor nu este o lectură de vacanță!" — ar putea să mă contrazică mulți dintre voi. Citiți, de exemplu "Legenda" propusă de Cătălin Frâncu și vă veți convinge că am dreptate.

- Nu v-am vorbit de cei pe care îi așteaptă confruntările cele mai palpitante: elevii din care s-a format *lotul național de informatică.* Ei abia acum intră într-un antrenament greu și variat. Cei 17 olimpici (numele lor le puteți citi în pagina 17 a revistei) vor primi probleme prin poștă electronică și tot așa vor transmite și rezolvările lor. După fiecare "pas" se va transmite situația punctajului, tot pe mail — de, suntem informaticieni în anul 1999! Între 31 mai și 6 iunie lotul se adună la Liceul de Informatică din Suceava, unde profesori chemați din întreaga țară vor încerca să le "dilateze" un pic creierele (dacă se mai poate, și eu cred că totdeauna se poate). Urmează o altă "bucățică" de confruntare pe e-mail. La sfârșitul lunii iulie se vor organiza (la Constanța) probele de selecționare a loturilor pentru *Olimpiada de Informatică a Europei Centrale și de Est* (CEOI) și pentru *Balcaniada de Informatică* (BOI). Să nu vă fie milă de membrii lotului, vor găsi timp și pentru o baie în mare sau un pic de plajă. Primul concurs menționat va avea loc la începutul lunii septembrie la *Brno* (Republica Cehă), Balcaniada anul acesta va fi la *Ioannina*, în Grecia, la sfârșitul lunii august. Nu, nu am uitat *Olimpiada Internațională* (*Inter*-ul), ea va fi organizată anul acesta în Turcia, la *Antalya*, în octombrie, așa că pentru selecția celor patru care vor reprezenta România la acest concurs, lotul se mai întâlnește o dată la Sinaia, unde se vor întrece pentru a ocupa un loc printre primii patru. Le dorim idei scânteietoare, putere de a se concentra și sperăm că succesul va fi pe măsura efortului.

Deci, cum am mai menționat, vacanța se apropie. Visele se amestecă deja cu proiecte de călătorie, cu planificarea unor excursii și întruniri de prieteni. Sper că toate aceste distracții nu vor însemna că ne uitați, dimpotrivă, veți aștepta cu *nerăbdare* începutul de an școlar și ... *GInfo.*

Clara Ionescu

Descoperiți
modul de lucru Visual

Acum vă puteți permite(*) un instrument de lucru original

(*) peste 70% reducere
numai pentru persoane fizice
numai pentru uz personal, acasă

Oferta este valabilă în perioada 01.04 - 30.06 1999

Denumirea produsului	Preț (USD)
VB Std 6.0 Win 32 English Intl CD Lrng	31
VB Pro 6.0 Win 32 Romanian CVUP CD	71
VB Pro 6.0 Win 32 Romanian CD	101
VC++ Std 6.0 Win 32 English Intl CD	31
VC++ Pro 6.0 Win 32 English Intl CVUP CD	71
VC++ Pro 6.0 Win 32 English Intl CD	101

Denumirea produsului	Preț (USD)
VFoxPro Pro 6.0 Win 32 English Intl CVUP CD	71
VFoxPro Pro 6.0 Win 32 English Intl CD	101
VInterDEV 6.0 Win 32 English Intl CVUP CD Pro	71
VInterDEV 6.0 Win 32 English Intl CD Pro	101
VJ++ Std 6.0 Win 32 English Intl CD	31
VJ++ Pro 6.0 Win 32 English Intl CVUP CD	71
VJ++ Pro 6.0 Win 32 English Intl CD	101

Prețul include 70% discount și se plătește în lei, la cursul de schimb al B.N.R. Prețul nu include T.V.A.

Pentru a beneficia de această ofertă,
contactați firma GeCAD, Microsoft Developer Dealer, tel.: 01-324 84 09,
http://www.gecad.ro/mspromo

PERSPECTIVA 2000

Microsoft®

PROBLEME propuse pentru REZOLVARE

Olimpiada Națională de Informatică, anul acesta a fost organizată la Mediaș. S-au acordat 11 premii I, 24 premii II, 33 premii III și 45 mențiuni. Mai ales pentru cei care nu au participat la olimpiadă, iată enunțurile problemelor!

CLASA IX

P059901: Expresie

prof. Marinel Șerban, Liceul de Informatică "Grigore Moisil", Iași

Prima linie a fișierului text **EX.IN** conține o expresie aritmetică fără paranteze având cel mult 255 caractere. Caracterele care pot să apară sunt:
- cifrele de la 0 la 9;
- literele mici a, b, c, d și e reprezentând variabile;
- operatorii aritmetici binari + și -

Pe următoarele linii din fișier pot să apară linii de forma: *<variabilă>=<întreg>*
reprezentând atribuiri de valori pentru variabilele care apar în expresie. Valorile care pot fi atribuite sunt cuprinse în intervalul (-1000000000,1000000000). O variabilă poate interveni într-o atribuire cel mult o dată.

Scopul programului pe care trebuie să-l scrieți este evaluarea și afișarea valorii expresiei pe prima linie a fișierului text **EX.OUT**.

Semnificația caracterelor care apar în expresie este cea obișnuită într-o expresie aritmetică. Orice succesiune de cifre formează o constantă naturală cuprinsă între 0 și 1000. Între o constantă și o variabilă consecutive, ca și între două variabile consecutive se consideră operația de înmulțire. Astfel, expresia 235aeae+c-12bb conține constantele 235 și 12 și are semnificația aritmetică:
235aeae+c-12bb

Dacă valoarea expresiei nu poate fi calculată, programul va afișa mesajul NEDEFINITA pe prima linie a fișierului de ieșire **EX.OUT**.

Exemple

EX.IN	EX.OUT
abc	NEDEFINITA
a=1	
b=2	

EX.IN	EX.OUT
235aeae+c-12bb	93981
b=1	
c=-7	
a=10	
e=2	

P059902: Nasturi

Stelian Ciurea, cadru didactic la Universitatea "Lucian Blaga", Sibiu

În fiecare pătrat al unei table de șah, de dimensiuni n×n, unde n este număr par, 3<n<31) avem câte un nasture. Știind că urmează să luăm x nasturi de pe tablă, programul vostru trebuie să precizeze dacă putem face această operație astfel încât, pe fiecare linie și pe fiecare coloană, să rămână un număr par de nasturi.

Date de intrare

Fișierul text **NASTURI.IN** conține pe prima linie numerele n și x separate print-un spațiu.

Date de ieșire

Rezultatele se vor scrie în fișierul text **NASTURI.OUT** care va avea formatul:
- dacă problema are soluție, pe linia i (i=1,2,...,n) a fișierului va fi descrisă linia i a tablei: pentru un pătrat cu nasture se va scrie litera 'o' iar pentru pătrat liber, caracterul '.' (punct). Între aceste caractere nu se vor afla spații sau alte caractere;
- dacă problema nu are soluție, pe prima linie a fișierului se va scrie textul 'fara solutie' (cu litere mici!).

Exemplu

NASTURI.IN	NASTURI.OUT
4 4	oo..
	oooo
	oo..
	oooo

P059903: Partide și coaliții

Valentin Gheorghiță, student, Universitatea Politehnică, București

În țara lui Papură Vodă s-au organizat alegeri anticipate la care participă n partide numerotate de la 1 la n ($n \leq 20$). La încheierea acestora, se cunoaște numărul de voturi obținute de fiecare partid. Pe baza acestor voturi se va calcula, pentru fiecare partid, "puterea" acestuia, defnită astfel: *"puterea" unui partid reprezintă numărul de coaliții nemajoritare în care pătrunzând, acestea devin majoritare.*

O coaliție poate fi formată din unul sau mai multe partide.

O coaliție de partide este majoritară atunci când totalul voturilor obținute de partidele ce o formează este strict mai mare decât jumătate din totalul voturilor exprimate la alegeri.

Cunoscându-se numărul n de partide și numărul de voturi obținute de fiecare dintre ele, să se determine "puterea" fiecărui partid.

Date de intrare

Fișierul text **PARTIDE.IN** are următorul format:

Pe prima linie este scris un număr natural n, reprezentând numărul de partide participante la alegeri.

Pe următoarea linie se găsesc n numere naturale separate printr-un spațiu:

$v_1\ v_2 \ldots v_n$ unde v_i ($0 \leq v_i \leq 3000$) reprezintă numărul de voturi obținut de partidul i ($1 \leq i \leq n$). Avem $\Sigma v_i \leq 5000$, adică totalul voturilor exprimate la alegeri nu depășește 5000.

Datele de ieșire:

În fișierul text **PARTIDE.OUT** pe o singură linie se vor scrie n numere: $p_1\ p_2 \ldots p_n$ unde p_i reprezentă puterea partidului i ($1 \leq i \leq n$); numerele vor fi despărțite între ele printr-un singur spațiu.

Exemplu

PARTIDE.IN	PARTIDE.OUT
3	3 1 1
2 1 1	

Pentru lămurirea rezultatului din exemplu specificăm că partidul 1 are puterea 3 deoarece poate intra în trei coaliții care vor deveni astfel majoritare:
- în coaliție cu partidul 2;
- în coaliție cu partidul 3;
- în coaliție cu coaliția formată din partidele 2 și 3.

P05994: Furtună în Balcani

Valentin Gheorghiță, student, Universitatea Politehnică, București

Taberele militare de pe teritoriul Yugoslaviei, amplasate în interiorul țării se pregătesc pentru un eventual atac terestru din partea forțelor NATO. Taberele nu ating marginea hărții și nu sunt lipite unele de altele.

Soldații sârbi trebuie să înconjoare toate taberele cu câte un câmp de mine plasate pe direcțiile N, S, E sau V față de sârma ghimpată ce împrejmuiește fiecare tabără.

Ajutați-i pe soldații sârbi să plaseze corect minele.

Date de intrare

Fișierul text **NATO.IN** conține harta taberelor sârbe sub forma unei matrice cu m linii și n coloane ($m, n \leq 100$), cu elemente caracterele '.' și 'X';
- '.' (punct) reprezintă teren (din interiorul sau exteriorul unei tabere);
- 'X' reprezintă sârma ghimpată care înconjoară taberele.

Pe prima linie a fișierului se găsesc dimensiunile matricei m și n, numere naturale separate printr-un spațiu. Pe următoarele m linii se găsește descrierea hărții:

```
c[1,1] c[1,2] ... c[1,n]
c[2,1] c[2,2] ... c[2,n]
...
c[m,1] c[m,2] ... c[m,n]
```

unde c[i,j] ($1 \leq i \leq m$, $1 \leq j \leq n$) sunt caractere din cele permise ('.' și 'X'). Prima și ultima linie, respectiv coloană, reprezintă marginea hărții, deci conțin numai caracterul '.'.

Obervație

Taberele sunt bine delimitate și conțin cel puțin un punct în interior.

Date de ieșire

Fișierul text **NATO.OUT** conține matricea de intrare completată cu caracterul M pe pozițiile de amplasare a minelor.

Exemplu

NATO.IN	NATO.OUT
5 11	.MMM...M...
.....	MXXXM.MXM..
.XXX...X...	MX.XXMX.XM.
.X.XX.X.X..	MXXXM.MXMXM
.XXX...X.X.	.MMM...M.M.
.....	

P059905: Semne

Prof. Eugen Ionescu, Liceul de Informatică "Tiberiu Popoviciu", Cluj

Fie n numere naturale consecutive, așezate unul după altul: 1,2,3,...,n. Să se scrie un program care stabilește semnul (+, - sau *) care trebuie pus în fața fiecărui element al șirului, astfel încât, evaluând expresia rezultată, să obținem valoarea 0. Există o singură excepție și anume în fața primului element trebuie pus fie +, fie -.

Ordinea în care se efectuează operațiile este conform regulilor aritmetice.

Date de intrare

Numărul n ($1 < n < 1000000$) se va citi de pe prima linie a fișierului text **ZERO.IN**.

Date de ieșire

Secvența de semne care trebuie scrise între numerele naturale 1,2,3,...,n se va scrie în fișierul text **ZERO.OUT**. Între oricare două semne (+, -, *) nu se va scrie nici număr, nici virgulă, nici spațiu.

În cazul în care nu se poate obține o expresie având valoarea 0, în fișier, pe prima linie se va scrie 'NU'.

Exemple:

ZERO.IN	**ZERO.OUT**
11	++--++--++-

(Echivalent cu +1+2-3-4+5+6-7-8+9+10-11=0)

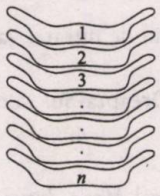
ZERO.IN	**ZERO.OUT**
5	-*++-

(Echivalent cu -1*2+3+4-5=0)

P059906: Teancul de farfurii

Prof. Dana Lica, Colegiul Național "I. L. Caragiale", Ploiești

În bucătăria unei gospodine se află un teanc de farfurii, care așteaptă să fie spălate. Gospodina care urmează să le spele este pusă în fața unei probleme: trebuie să spele un număr cât mai mare de farfurii într-un anumit timp. Ea cunoaște câte minute durează spălarea fiecărei farfurii; de asemenea ea poate să renunțe la spălarea ei și să o spargă, spargerea necesitând exact un minut, pentru orice farfurie.



Farfuriile sunt numerotate cu numere de la 1 la n, începând cu farfuria din vârful teancului spre cea de la baza teancului, ca în figura alăturată.

Farfuria care poate fi spălată la un moment dat este cea situată în vârful teancului. Inițial gospodina poate spăla sau sparge farfuria cu numărul 1, după care poate spăla sau sparge cu farfuria numărul 2, ș.a.m.d. Gospodina va trebui să decidă dacă spală o anumită farfurie sau o sparge, știind că are la dispoziție un timp t exprimat în minute.

Determinați care este numărul maxim nmax de farfurii pe care le poate spăla gospodina în timpul t dat.

Determinați de asemenea și timpul minim tmin (tmin≤t) în care se spală cele nmax farfurii.

Date de intrare

Fișierul text **FARFURII.IN** are următorul format:

Pe prima linie sunt scrise două numere naturale separate printr-un spațiu: n t, reprezentând numărul de farfurii, (n≤75) respectiv timpul t (t≤1000) pe care îl are la dispoziție gospodina.

Pe următoarea linie se găsesc n numere naturale separate printr-un spațiu: t_1 t_2 ... t_n, t_i≤100, reprezentând timpul de spălare al farfuriei i, în minute, (1≤i≤n).

Date de ieșire

Rezultatele (nmax tmin) se vor scrie în fișierul text **FARFURII.OUT** pe o singură linie. Cele două numere reprezintă numărul maxim de farfurii spălate și timpul efectiv de spălare (tmin≤t).

Exemplu

FARFURII.IN	**FARFURII.OUT**
4 4	2 3
1 4 1 3	

CLASA X

P059907: Gaze la vile

Prof. Ioan Maxim, Liceul de Informatică, Suceava

Proprietarii a n (2≤n≤50) vile din zona rezidențială a municipiului Mediaș au solicitat Regiei ROMGAZ instalarea de cisterne cu metan. ROMGAZ dispune de cisterne care asigură consumul pentru k vile pe o perioadă acceptabilă de timp (k este divizor al lui n, 2≤k≤n).

Pentru a i se asigura securitatea, fiecare cisternă este instalată lângă o vilă. Costul conectării unei vile este egal cu lungimea conductei până la cisternă sau până la o altă vilă conectată.

Să se determine grupe de k vile astfel încât costul conectării tuturor vilelor să fie minim.

Observații

Există vile între care nu se poate instala conductă de conectare. Distanța între acestea nu este specificată. Costul conectării vilei de lângă cisternă este nul.

Date de intrare

Fișierul de intrare **GAZE.IN** conține pe prima linie:

n k - numărul total de vile și numărul de vile care pot fi conectate la o cisternă, urmată de un număr de linii de forma:

i j_1 d_1 j_2 d_2 ... j_m d_m - vila i se poate conecta la una dintre vilele $j_1, j_2, \ldots, j_m$, și se află respectiv, la distanța $d_1, d_2, \ldots, d_m$ de acestea ($ij_1, ij_2, \ldots, ij_m$). Distanțele $d_1, d_2, \ldots, d_m$ sunt numere naturale ($1 \le d_1, d_2, \ldots, d_m \le 50$).

Datele de ieșire

Fișierul de ieșire **GAZE.OUT** conține pe prima linie:

t - costul total al conectării celor n vile, urmat de (n **div** k) linii de forma:

j_1 j_2 ... j_k c - vilele j_1 j_2 ... j_k sunt conectate la aceeași cisternă, iar costul conectării lor este c, dacă cele n vile pot fi conectate câte k și

0 - în situația în care problema nu are soluție.

Exemple

GAZE.IN	**GAZE.OUT**
9 3	20
1 6 1 8 3 9 5	2 4 6 6
2 4 4 6 2	1 8 9 8
3 5 3 7 3	3 5 7 6
4 9 6	
5 7 3	

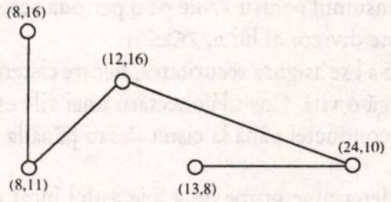
GAZE.IN
```
4 2
1 2 1 3 1 4 1
```

GAZE.OUT
```
0
```

P059908: Rețea

Prof Dorin Mânz, Liceul de Informatică, Timișoara

Se dorește legarea în rețea a n calculatoare (n∈[2,100]). Problema constă în obținerea unei rețele liniare în care fiecare calculator este legat exact de alte două calculatoare, exceptând calculatoarele din capete, care au o singură legătură. De exemplu, în desenul de mai jos calculatoarele sunt identificate prin coordonatele lor (relativ la un sistem de axe neprecizat).



Scopul problemei constă în minimizarea lungimii cablului folosit; se cere deci un mod optim de reconectare al calculatoarelor în linie, astfel încât cablul folosit să aibă lungimea minimă. Necesarul de cablu pentru a lega direct două calculatoare este egal cu distanța dintre acestea, la care se adaugă 10 metri de cablu, utilizați pentru conexiuni anexe.

Date de intrare

Fișierul de intrare cu numele **RETEA.IN** va conține pe prima linie un singur număr n (numărul de calculatoare). În continuare, pe linii succesive, se dau coordonatele fiecărui calculator sub forma unei perechi de numere întregi (din intervalul [0,100]). Pentru exemplul prezentat avem:

```
5
8  11
8  16
12 16
13  8
24 10
```

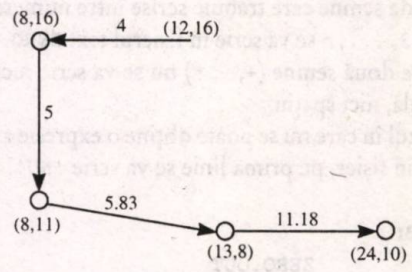
Date de ieșire

Ieșirea constă din n linii. Pe prima linie a fișierului **RETEA.OUT** se va scrie lungimea totală a cablului. Fiecare din celelalte linii este formată din trei numere l, s și e, unde: l este lungimea cablului necesar legării directe a calculatoarelor date la intrare pe liniile s respectiv e. Ordinea de parcurgere a rețelei este arbitrară.

Pentru rețeaua optimă din exemplu, ieșirea va fi:

```
66.01
14 3 2
15 2 1
15.83 1 4
21.18 4 5
```

Acest mod de conectare corespunde figurii de mai jos:



P059909: Zone

Prof. Adrian Niță, Liceul "Emanuil Gojdu", Oradea

Considerăm un set de n puncte, 0<n≤100.

Centrul Geometric este un punct care are coordonatele x și y, media aritmetică a absciselor, iar y este media aritmetică a ordonatelor tuturor punctelor.

Se definesc următoarele noțiuni:

- *Dreptunghi* este cel mai mic dreptunghi care are laturile paralele cu axele de coordonate și care include toate punctele date;
- *Zona* este un cerc cu centru în *Centrul Geometric* care este inclus în *Dreptunghi*;
- *Zona Moartă* este cea mai mare *Zonă* care nu conține nici un punct, din cele n date, în interiorul său;
- *Mega Zona* este cea mai mare *Zonă*.

Pentru fiecare set de puncte din fișierul de intrare, va trebui *să determinați*:

- coordonatele colțurilor Stânga-Jos, Dreapta-Sus ale *Dreptunghi*-ului;
- diametrul *Zonei Moarte*;
- procentul punctelor care sunt în interiorul *Mega Zonei*. Se consideră că punctele de pe circumferința cercului fac parte din interior.

Date de intrare

Fișierul **ZONE.IN** conține n+1 linii. Pe prima linie este scris numărul n, reprezentând numărul de puncte, iar pe fiecare din următoarele n linii sunt scrise coordonatele reale x, y ale celor n puncte separate prin spații.

Date de ieșire

Rezultatele, (numere reale cu două zecimale, obținute prin rotunjire) se scriu în fișierul **ZONE.OUT**, care are structura:

- pe liniile unu și doi se vor scrie coordonatele *Dreptunghi*-ului, coordonatele x, y pentru Stânga-Jos și respectiv Dreapta-Sus;
- pe a treia linie se vor scrie coordonatele *Centrului Geometric;*
- pe a patra linie se scrie diametrul *Zonei Moarte* și procentul de puncte din *Mega Zonă*.

Pe fiecare linie, numerele vor fi separate prin câte un spațiu.

Exemplu

ZONE.IN	ZONE.OUT
10	-2.00 -1.00
-1.6 1.8	2.00 2.00
1 2	0.00 0.40
2 0.6	0.80 40.00
1.6 -1	
-1 -1	
-2 0	
-1 1	
0 1	
1 -0.4	
0 0	

P059910: Seamănă, dar nu răsare

Prof. Dorina Popa, Liceul "Mihail Eminescu", Botoşani

Un graf ***neorientat*** are proprietatea că gradul oricărui vârf este mai mare decât jumătatea numărului vârfurilor. Graful se dă prin matricea sa de adiacenţă. Afişaţi un ciclu hamiltonian al grafului dat. Dacă graful nu are nici un ciclu hamiltonian, se va afişa mesajul: SEAMANA, DAR NU RASARE.

Date de intrare

Fişierul de intrare **HAM.IN** conţine pe prima linie:

n - numărul vârfurilor grafului ($3 \le n \le 100$), urmat de n linii de forma:

a_{i1} a_{i2} ... a_{in} - cu semnificaţia $a_{ij}=1$ dacă nodurile i şi j sunt adiacente şi $a_{ij}=0$ în caz contrar.

Date de ieşire

Fişierul de ieşire **HAM.OUT** conţine pe o singură linie vârfurile ciclului hamiltonian, în ordinea în care acestea apar în ciclu, separate printr-un spaţiu. Se cere o singură soluţie.

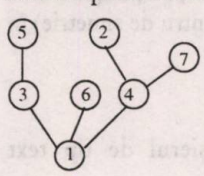
Exemplu

HAM.IN	HAM.OUT
5	1 5 2 3 4
0 1 1 1 1	
1 0 1 1 1	
1 1 0 1 1	
1 1 1 0 0	
1 1 1 0 0	

P059911: La cules de zmeurică

Cătălin Frâncu, student, Universitatea Politehnică, Bucureşti

O colonie de termite a ieşit la cules de zmeurică în tufa din apropierea mușuroiului. Consiliul Economic al Mușuroiului a hotărât că această tufă trebuie să asigure hrana mușuroiului atât anul acesta, cât şi anul următor, după aceasta ea nemaiprezentând interes.



Tufa de zmeurică are un aspect arborescent: din pământ pornesc, una sau mai multe ramuri, care la rândul lor se ramifică în mai multe rămurele etc. În fiecare an, în vârful fiecărei crenguţe care nu se mai ramifică, creşte câte un bob de zmeură. Pentru buna desfăşurare a culesului, Consiliul de Supervizare a etichetat îmbinările rămurelelor şi vârfurile crenguţelor cu numere naturale consecutive, începând de la 1. Rădăcina tufei (cea de la contactul cu solul) are numărul 1. În figura din stânga, fructele de zmeură cresc în nodurile 2, 5, 6 şi 7. Bugetul mușuroiului se modifică astfel în timpul culesului:

1) Fiecare fruct cules aduce P unităţi de hrană în hambar.
2) Culegerea unui fruct "en detail" înseamnă trimiterea unei termite de la sol până la acel fruct pentru a-l tăia de pe crenguţa lui. Termita consumă câte C unităţi de hrană din hambar pentru fiecare rămurică pe care o parcurge până la bobul de zmeură. În figura de mai sus, culegerea fructelor 2, 5 şi 7 necesită consumarea a 2C unităţi de hrană, iar culegerea fructului 6 necesită C unităţi de hrană. Anul viitor, un nou bob va creşte în locul celui cules.
3) Zmeurica poate fi culeasă şi "en-gros", producând însă daune tufei. Este posibil ca un nod întreg să fie retezat, împreună cu tot subarborele său. Pentru aceasta va fi trimisă o termită de la sol până la nodul care va fi retezat, termita consumând câte C unităţi de hrană pentru fiecare ramură parcursă. După aceasta, fructele de pe creanga retezată cad pe sol şi sunt culese cu cost 0. În figură, retezarea nodului 4 se face cu cost 1C şi aduce un profit de 2P (datorită fructelor din nodurile 2 şi 7). Dezavantajul este că anul viitor în nodul retezat va creşte o singură zmeurică. Dacă nodul 1 este retezat (tufa este tăiată din rădăcină), anul viitor nu va mai creşte nimic.

Se cere să se planifice o manieră de culegere a zmeurei în primul şi în al doilea an, astfel încât în final hambarul să fie cât mai îmbelşugat.

Date de intrare

Fişierul text **ZMEURICA.IN** conţine pe prima linie valorile N, P şi C ($2 \le N \le 100$, $1 \le P$, $C \le 1000$), separate prin spaţii. N este numărul total de noduri etichetate, iar P şi C sunt numere întregi. Pe următoarele N linii se află descrierea tufei de zmeură. Linia k+1 conţine numărul de noduri care se află imediat deasupra nodului k în tufă, apoi valorile acestor noduri în ordine crescătoare. Pentru figura de mai sus fişierul este:

```
7 5 1     % N=7, P=5, C=1
3 3 4 6   % După nodul 1 (rădăcina) urmează trei
            noduri, şi anume 3, 4 şi 6
0         % După nodul 2 nu mai urmează nimic
1 5       % După nodul 3 urmează un nod, şi anume 5
2 2 7     % După nodul 4 urmează două noduri, şi
            anume 2 şi 7
0         % După nodul 5 nu mai urmează nimic
0         % După nodul 6 nu mai urmează nimic
0         % După nodul 7 nu mai urmează nimic
```

Date de ieşire

Fişierul text **ZMEURICA.OUT** trebuie să conţină trei linii. Pe prima se va afla cantitatea maximă de hrană (venit minus cheltuieli) care poate fi acumulată din exploatarea tufei de zmeurică. Pe linia a doua se vor indica, separate prin spaţii, în orice ordine, numerele nodurilor unde se vor face tăieturi în primul an (pentru a tăia fie un fruct, fie o creangă întreagă). Pe linia a treia se vor indica nodurile unde se fac tăieturi în al doilea an. Dacă în al doilea an nu mai există nici un nod de tăiat, linia a treia va fi lăsată goală. Pentru exemplul dat, ieşirea poate fi:

```
34
3 6 2 7
1
```

Într-adevăr, în anul I retezăm rămurica nodul 3, culegând fructul 5 (cost 1, venit 5), şi culegem fructele 6 (cost 1, venit 5), 2 (cost 2, venit 5) şi 7 (cost 2, venit 5). În anul II vor creşte fructe în nodurile 3, 6, 2 şi 7, pe care le culegem "en gros", tăind tufa din rădăcină (cost 0, venit 54). În total avem costul 6 şi venitul 40, deci profitul net este 34, maximul care se poate "stoarce" din tufa dată.

P059912: Degrade

Prof. Ioan Maxim, Liceul de Informatică, Suceava

Un tablou pătratic, de dimensiuni $n \times n$ ($2 \le n \le 150$) conţine numere naturale din intervalul $[0, 99]$. Aceste numere reprezintă nuanţe de "*gri*" (0 este alb, 99 este negru).

Se cere ca numai prin interschimbări de elemente alăturate (pe linie, coloană sau diagonală), să se realizeze în tablou un degrade cu raza pe diagonala principală (nuanţa cea mai apropiată de alb va fi în colţul din stânga sus, iar cea mai apropiată de negru în colţul din dreapta jos).

Pe diagonala secundară şi pe orice paralelă la aceasta, nuanţele mai închise sunt dispuse spre exterior, adică $a_{i,j} \le \min(a_{i,j+1}, a_{i+1,j}) \le \max(a_{i,j+1}, a_{i+1,j}) \le a_{i+1,j+1}$ pentru orice $i,j=1,n-1$ şi $a_{i,j+1} \ge a_{i+1,j}$ pentru $i \le j$ şi $a_{i,j+1} \le a_{i+1,j}$ pentru $i < j$.

Date de intrare

Fişierul de intrare **DEGRADE.IN** conţine n linii cu câte n elemente, separate prin câte un spaţiu.

Datele de ieşire

Fişierul de ieşire **DEGRADE.OUT** va conţine tabloul prelucrat (tot n linii cu câte n numere separate printr-un spaţiu).

Exemple

DEGRADE.IN

```
4 1
2 3
```

DEGRADE.OUT

```
1 3
2 4
```

DEGRADE.IN

```
4 1 2 3
2 3 9 5
3 4 6 7
5 6 9 4
```

DEGRADE.OUT

```
1 2 3 5
2 3 4 7
3 4 4 9
5 6 6 9
```

CLASA XI-XII

P059913: Formă normală

Prof. Emanuela Cerchez, Liceul de Informatică "Grigore Moisil", Iaşi

Fie E o expresie care respectă următoarele condiţii:

- poate conţine numai operatorii + (adunare) şi * (înmulţire);
- operanzii pot fi doar nume de variabile formate dintr-o singură literă mică a alfabetului englez;
- poate conţine paranteze rotunde.

Ştiind că puteţi aplica următoarele proprietăţi:

1. Comutativitatea adunării: $a+b=b+a, \forall a,b$.
1'. Comutativitatea înmulţirii: $a*b=b*a, \forall a,b$.
2. Asociativitatea adunării: $a+(b+c)=(a+b)+c = a+b+c, \forall a,b,c$.
2'. Asociativitatea înmulţirii: $a*(b*c)=(a*b)*c=a*b*c, \forall a,b,c$.
3. Distributivitatea înmulţirii faţă de adunare: $a*(b+c)=a*b+a*c, \forall a,b,c$.

Aduceţi expresia dată în formă normală.

***Forma normală a unei expresii** este o sumă de produse, echivalentă cu expresia dată, astfel încât atât termenii sumei, cât şi factorii din fiecare termen al sumei sunt ordonaţi alfabetic.*

Date de intrare

Fişierul de intrare **EXP.IN** conţine pe prima linie o expresie având cel mult 100 de caractere, corectă în raport cu restricţiile enunţate mai sus. Expresia nu conţine spaţii.

Date de ieşire

Expresia în formă normală (având lungimea maximă de 255 de caractere) va fi afişată pe prima linie a fişierului de ieşire **EXP.OUT**, fără spaţii.

Exemplu

EXP.IN

```
x*a+c*(a+b*c*d)
```

EXP.OUT

```
a*c+a*x+b*c*c*d
```

P059914: Împărţirea poligonului

Prof. Doru Popescu Anastasiu, Liceul "Radu Greceanu", Slatina

Se dă un poligon convex cu n vârfuri ($4 \le n \le 50$) prin coordonatele vârfurilor (numere întregi din intervalul $[-2000, 2000]$) date în sensul acelor de ceasornic.

a) Să se verifice dacă poligonul are centru de simetrie. Punctul S este centru de simetrie dacă simetricul oricărui punct de pe poligon faţă de S aparţine poligonului.

b) Să se împartă poligonul (dacă are centru de simetrie) în paralelograme.

Date de intrare

Datele de intrare se citesc din fişierul de tip text **POLIGON.IN** având structura:

```
n
x1  y1
x2  y2
...
xn  yn
```

Date de ieşire

Datele de ieşire se vor scrie în fişierul de tip text **POLIGON.OUT** având structura:

mesaj

```
px11  py11  px21  py21  px31  py31  px41  py41
px12  py12  px22  py22  px32  py32  px42  py42
...
px1k  py1k  px2k  py2k  px3k  py3k  px4k  py4k
```

unde *mesaj* poate fi DA sau NU, în funcţie de existenţa centrului de simetrie.

- dacă mesajul este 'NU', fişierul va conţine doar prima linie;
- dacă mesajul este 'DA', fişierul va conţine în plus atâtea linii câte paralelograme au rezultat în urma împărţirii poligonului iniţial;
- un paralelogram este descris prin coordonatele vârfurilor sale în sensul acelor de ceasornic.

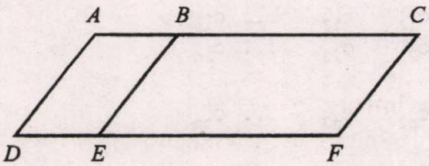
Exemple

POLIGON.IN	POLIGON.OUT
8	DA
20 30	20 0 10 10 20 10 30 0
30 30	10 10 10 20 20 20 20 10
40 20	10 20 20 30 30 30 20 20
40 10	20 10 20 20 30 10 30 0
30 0	20 20 30 30 40 20 30 10
20 0	40 20 40 10 30 0 30 10
10 10	
10 20	

POLIGON.IN	POLIGON.IN
4	NU
10 20	
40 30	
40 10	
20 10	

Restricţie:

Dacă două paralelograme cu interioarele disjuncte au o latură comună şi alte două laturi în prelungire (ca în figura alăturată paralelogramele *ABED* şi *BCFE*) se va afişa un singur paralelogram (în figură *ACFD*).



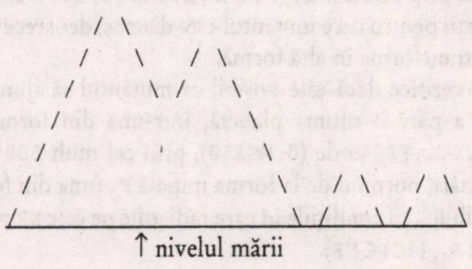
P059915: Peisaje

Lect. univ. Clara Ionescu, Universitatea "Babeş Bolyai", Cluj

Linia orizontului dintr-un peisaj se poate desena pe un caroiaj, cu ajutorul caracterelor '/' şi '\', aceasta conturându-se într-un "zig-zag" obişnuit. *O linie a orizontului este corectă dacă este desenată cu acelaşi număr de caractere '/', respectiv '\' şi fiecare coloană a caroiajului conţine un singur caracter.* De asemenea, această linie *trebuie să pornească dintr-o celulă a caroiajului aflată la nivelul mării şi se termină tot într-o celulă aflată la nivelul mării.* Pe parcursul trasării, linia *nu coboară niciodată sub nivelul mării.* Linia orizontului începe cu caracterul '/' şi se termină cu caracterul '\'. După caracterul '/' poate fi plasat pe coloana următoare caracterul '/' pe linia superioară sau caracterul '\' pe aceeaşi linie. Analog, după caracterul '\' poate fi plasat pe coloana următoare caracterul '\' pe linia inferioară sau caracterul '/' pe aceeaşi linie.

Ştiind că într-o celulă a caroiajului se desenează un singur caracter, definim *înălţimea unui munte* ca fiind egală cu numărul liniilor din caroiaj dintre nivelul mării şi vârful muntelui (muntele /\ are înălţimea 1). *Vârful muntelui* se formează din două caractere /\ situate pe aceeaşi linie şi pe coloane alăturate.

Scrieţi un program care calculează numărul de posibilităţi de a trasa linii ale orizontului cu un număr precizat de perechi de caractere '/' şi '\', astfel încât cel puţin un munte are o înălţime mai mare sau egală cu o anumită înălţime dată.



Date de intrare

Din fişierul de tip text **PEISAJ.IN** se vor citi două numere naturale n şi k (1≤n≤31, reprezentând numărul caracterelor '/', (acesta fiind acelaşi cu numărul caracterelor '\'). Numărul k (1≤k≤n) reprezintă înălţimea minimă a cel puţin unui munte.

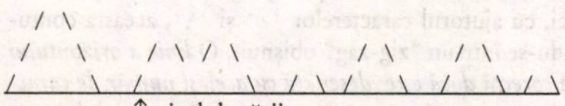
Date de ieşire

În fişierul de tip text **PEISAJ.OUT** se va scrie numărul liniilor de orizont "cu munţi", ce pot fi desenate cu n perechi de caractere ('/', respectiv '\'), unde cel puţin un munte are o înălţime cel puţin egală cu k.

Exemplu

PEISAJ.IN	PEISAJ.OUT
3 2	4

Corespunzător exemplului avem următoarele linii de orizont posibile:



P059916: Mutantul călător

Prof. Roxana Tâmplaru, Liceul de Informatică, Craiova

Se consideră NP planete (0<NP≤60). Fiecare planetă este împărțită în două emisfere: una radiată (nivelul radiațiilor fiind un număr întreg din intervalul [1,1000]), și una neradiată (nivelul radiației este 0).

Un mutant vizitează pe rând toate planetele, în ordinea 1,2,...,NP, fiecare planetă fiind vizitată o singură dată.

Mutantul poate veni pe o planetă în oricare dintre emisfere și poate părăsi planeta prin oricare dintre ele.

De asemenea, el poate trece dintr-o emisferă în alta.

Pentru fiecare dintre planete, mutantul trebuie să treacă o singură dată prin emisfera radiată, și de zero sau mai multe ori prin cea neradiată.

Pe fiecare planetă, în emisfera radiată, mutantul suferă o singură transformare, iar în cea neradiată suferă una sau mai multe transformări succesive. În urma unei transformări, acesta poate lua una din formele posibile, în funcție de nivelul radiației și de forma sa curentă. Mutantul poate avea cel mult 250 forme distincte ($F_1, F_2, \ldots F_{NF}$), care se reprezintă prin valori întregi din intervalul [0,249]. Există și radiații pentru care mutantul este distrus, deoarece nu se poate transforma în altă formă.

Să se verifice dacă este posibil ca mutantul să ajungă, după ce a părăsit ultima planetă, într-una din formele $FF_1, FF_2, \ldots, FF_M$ unde (0<M≤250), prin cel mult 500 de transformări, pornind de la forma inițială F_0 (una din formele posibile), în condițiile în care radiațiile pe cele NP planete sunt R_i, ($1 \le i \le NP$).

În cazul în care problema poate fi soluționată se va afișa o singură soluție.

Date de intrare

Fișierul **MUTANT.IN** are următorul conținut:

Pe prima linie: NP

Pe a doua linie: NF F_1 F_2 ... F_{NF}

Pe linia a treia: F_0

Pe linia a patra: R_1 R_2 ... R_{NP}

Pe linia a cincea: M FF_1 FF_2 ... F_M

În continuare, până la sfârșitul fișierului, pe fiecare linie se află o transformare posibilă: F R A_1 A_2 ... unde F este forma curentă, R nivelul radiației, iar A_i sunt formele posibile în care se poate trece din F.

Fișierul conține doar transformările pentru care radiația nu distruge mutantul. Numărul maxim de transformări este 500. Fișierul conține doar numere întregi; cele care se află pe aceeași linie sunt separate printr-un spațiu.

Date de ieșire

Fișierul **MUTANT.OUT** va avea structura:

a) dacă problema are soluție:

Pe prima linie: DA

Pe a doua linie: B_1 C_1 B_2 C_2 ... B_P

unde $B_1, \ldots, B_P$ este succesiunea de forme pentru care:

$B_1 = F_0$

B_P este una din formele $FF_1, \ldots, FF_M$

Din forma B_i mutantul trece în forma B_{i+1} printr-o singură transformare datorată unei radiații C_i, (zero sau diferită de zero) unde $1 \le i < p$.

b) dacă problema nu are soluție:

Pe prima linie: NU

Exemplu

MUTANT.IN	MUTANT.OUT
2	DA
7 0 1 2 3 4 5 6	0 0 1 1 5 2 6
0	
1 2	
2 5 6	
0 0 1 2	
0 1 3	
0 2 4	
1 1 5	
2 0 1	
3 1 5	
5 2 6	

P059917: Oli' la Mediaș...

Prof. Maria Nița, Liceul "Emanuil Gojdu", Oradea

În vederea organizării Olimpiadei de Informatică, n firme de calculatoare doresc să trimită la Mediaș echipamente de tehnică de calcul. Organizatorii au nevoie de n tipuri de echipamente, numerotate de la 1 la n. Transportul echipamentelor presupune anumite cheltuieli, pe care fiecare firmă le prezintă organizatorilor. Știind că toate firmele pot acoperi fără probleme necesarul de tehnică de calcul și că fiecare firmă dispune de toate tipurile de tehnică de calcul, ajutați organizatorii să asigure dotarea tehnică necesară concursului, astfel încât:

- de la fiecare firmă se aduce un singur transport, conținând un singur tip de tehnică de calcul;
- cheltuielile totale pentru realizarea dotării pentru concurs să fie minime.

Date de intrare

Datele de intrare se citesc din fișierul **MEDIAS.IN**, având structura:

n

c_{11} c_{12} c_{13} ... c_{1n}

c_{21} c_{22} c_{23} ... c_{2n}

...

c_{n1} c_{n2} c_{n3} ... c_{nn}

unde:
- n reprezintă numărul de firme, același cu numărul de tipuri de tehnică de calcul (0<n<100);
- c_{ij} reprezintă cheltuielile pentru transportul în Mediaș al tehnicii de calcul de tip i de la firma j (c_{ij} sunt numere naturale din intervalul [1,255]).

Date de ieșire

Rezultatele se scriu în fișierul **MEDIAS.OUT** sub forma:

$1\ i_1\ c_{1i_1}$
$2\ i_2\ c_{2i_2}$
...
$n\ i_n\ c_{ni_n}$
min

unde:
- k reprezintă tipul de tehnică de calcul;
- i_k reprezintă firma de la care se aduce tehnica de calcul având tipul k;
- c_{ki_k} reprezintă cheltuielile de transport al tehnicii de tip k de la firma i_k, ($1 \le k \le n$);
- min reprezintă cheltuielile totale necesare pentru dotare.

Exemplu

MEDIAS.IN	MEDIAS.OUT
6	1 4 14
17 43 27 14 39 52	2 6 13
29 24 69 90 23 13	3 5 16
18 90 62 12 16 70	4 2 14
58 14 6 18 73 64	5 1 15
15 41 38 36 40 60	6 3 18
25 44 18 44 13 50	90

P059918: Telescaun

Prof. Emanuela Cerchez, Liceul de Informatică "Grigore Moisil", Iași

Un fermier montan dorește să construiască un telescaun în ferma proprie. În acest scop a prospectat terenul și a identificat o zonă de formă dreptunghiulară de maxim interes turistic. A construit apoi o hartă a zonei, a împărțit-o în pătrățele de latură 1 și a înscris în fiecare pătrățel cota porțiunii de teren corespunzătoare. Telescaunul va avea un traseu rectiliniu, care trebuie să unească un punct de cotă maximă (considerat în centrul pătrățelului respectiv) cu un punct de cotă minimă (considerat, de asemenea, în centrul pătrățelului respectiv). Pentru a nu face investiții suplimentare, cotele de-a lungul traseului trebuie să fie în ordine descrescătoare. În cazul în care există mai multe posibilități, fermierul va prefera, evident, un traseu de lungime minimă.

Exemple

5	2	2	9
2	2	6	8
3	2	4	4
2	1	2	2

Traseul de la maxim la minim parcurge cotele 9,8,6,4,2,1 și este soluție.

5	2	2	9
2	2	6	8
3	4	2	4
2	1	2	2

Traseul de la maxim la minim parcurge cotele 9,8,6,2,4,1, deci nu este soluție.

5	2	9	2
2	5	6	8
3	4	2	4
2	1	2	2

Traseul de la punctul de maxim la cel de minim parcurge cotele 9,6,4,1 deci este soluție.

Date de intrare

Datele de intrare se citesc din fișierul **COTE.IN** care are următoarea structură:

n m
$c_{11}\ c_{12}\ \dots\ c_{1m}$
$c_{21}\ c_{22}\ \dots\ c_{2m}$
...
$c_{n1}\ c_{n2}\ \dots\ c_{nm}$

cu următoarea semnificație:
n = lungimea hărții ($n \in N^*$, $n \le 100$),
m = lățimea hărții ($m \in N^*$, $m \le 100$),
c_{ij} = cota porțiunii de teren de coordonate i, j, ($c_{i,j} \in N$, $c_{i,j} \le 6000$).

Date de ieșire

Dacă nu se poate găsi nici un traseu care să corespundă cerințelor, fișierul de ieșire **COTE.OUT** conține, pe prima linie mesajul IMPOSIBIL, sau, în cazul în care un asemenea traseu există, va conține:

L
$x_M\ y_M\ x_m\ y_m$

unde:
- L este număr real cu 2 zecimale (obținut prin rotunjire) și reprezintă lungimea traseului pe hartă;
- $x_M\ y_M$ sunt coordonatele punctului de cotă maximă;
- $x_m\ y_m$ sunt coordonatele punctului de cotă minimă.

Exemplu

COTE.IN	COTE.OUT
4 4	3.61
1 2 2 2	3 4 1 1
2 2 3 3	
3 3 3 4	
2 3 2 2	

BARAJ

P059919: Expoziție

Lect. univ. Clara Ionescu, Universitatea "Babeș Bolyai", Cluj

Într-o sală de expoziție se vor expune aparate de tehnică de calcul (se cunosc coordonatele lor în plan). În punctul de coordonate (0,0) se află o sursă de curent. Pentru alimentarea aparatelor, acestea vor fi legate cu cablu direct la sursa de curent, sau la aparate deja alimentate (direct sau

indirect). Lungimea unui cablu este egală cu distanța în linie dreaptă între punctele pe care le conectează.

Organizatorii au inițiat un concurs pentru a stabili lungimea totală minimă a cablului necesar pentru alimentarea aparatelor. Se vor acorda trei premii pentru primele trei soluții diferite, în ordinea crescătoare a lungimii cablurilor. Trebuie respectate următoarele cerințe:

1) toate aparatele trebuie alimentate;
2) două soluții sunt considerate diferite dacă una dintre ele conține cel puțin două aparate conectate direct care în cealaltă soluție nu sunt conectate direct.

Date de intrare

Datele de intrare se citesc din fișierul de tip text **EXPO.IN**. Fișierul are următoarea structură:

Pe prima linie este scris numărul întreg n ($3 \le n \le 50$), reprezentând numărul aparatelor care necesită alimentare.

Pe următoarele n linii sunt scrise perechi de numere reale reprezentând coordonatele aparatelor amplasate în interiorul sălii:

```
x1 y1   // coordonatele aparatului 1
x2 y2   // coordonatele aparatului 2
...
xn yn   // coordonatele aparatului n
```

Sursa de energie este considerată "aparatul" având numărul de ordine 0.

Date de ieșire

Rezultatele se vor scrie în fișierul **EXPO.OUT**.

Cele trei soluții de alimentare a aparatelor vor forma trei seturi de date, separate prin câte o linie vidă.

Un set este format din mai multe linii, unde pe o linie se specifică aparatele legate printr-un cablu având lungimea egală cu distanța dintre cele două aparate. Un aparat se identifică prin numărul său de ordine. Numerele de pe aceeași linie se vor separa printr-un singur spațiu. În cadrul unui set ordinea perechilor de numere este oarecare. De asemenea, nu contează ordinea în care sunt scrise numerele într-o pereche.

Exemplu

EXPO.IN	EXPO.OUT
3	0 2
2.2 2.2	1 2
1.1 1.1	1 3
3.3 3.3	
	0 1
	1 2
	1 3
	0 1
	2 0
	1 3

P059920: Camere luminate

Valentin Gheorghiță, student, Universitatea Politehnică, București

Într-o vilă sunt n camere ($1 \le n \le 11$) numerotate cu numere de la 1 la n, fiecare fiind luminate cu ajutorul unui singur bec. Pentru fiecare cameră k ($1 \le k \le n$) se cunoaște numărul camerelor vecine (nr_k) precum și care sunt aceste camere. Două camere sunt vecine, dacă se poate trece din una în cealaltă direct.

În fiecare cameră k ($1 \le k \le n$) din vilă se găsesc două grupuri de întrerupătoare:

- un grup format din $nr1_k$ întrerupătoare pentru aprinderea luminilor în anumite camere;
- un grup format din $nr2_k$ întrerupătoare pentru stingerea luminilor în anumite camere.

Proprietarul vilei se întoarce seara târziu acasă și observă că singura cameră luminată este camera 1. El intră în casă numai prin camera 1 și dorește să ajungă în camera n, care este dormitorul său. Superstițios, el nu va intra niciodată într-o cameră neluminată, dar ajuns în final, în dormitor nu va putea *dormi liniștit* dacă în oricare din celelalte camere, în afara dormitorului, lumina este aprinsă.

Durata unui drum dintre camera 1 și dormitor este egală cu **suma** *dintre numărul de deplasări dintr-o cameră în alta și numărul total de aprinderi și stingeri ale luminii, începând cu momentul în care proprietarul se află în camera 1 și până ajunge în dormitor.*

Deoarece seara când vine acasă este foarte obosit, el dorește să determine un drum de *durată minimă* de la camera 1 la dormitor (camera n) astfel încât el să poată dormi liniștit. Prin orice cameră (inclusiv prin dormitor) se poate trece de mai multe ori.

Date de intrare

Fișierul **VILA.IN** are următorul format:

```
n
nr1  vec1[1] vec1[2] ... vec1[nr1]
nr2  vec2[1] vec2[2] ... vec2[nr2]
...
nrn  vecn[1] vecn[2] ... vecn[nrn]
nrl1 ap1[1] ap1[2] ... ap1[nrl1]
nrl2 ap2[1] ap2[2] ... ap2[nrl2]
...
nrln apn[1] apn[2] ... apn[nrln]
nr21 st1[1] st1[2] ... st1[nr21]
nr22 st2[1] st2[2] ... st2[nr22]
...
nr2n stn[1] stn[2] ... stn[nrln]
```

- n reprezintă numărul camerelor din vilă;
- nr_k reprezintă numărul camerelor vecine cu camera k;
- $vec_k[j]$ reprezintă cea de a j-a cameră vecină cu camera k;
- nrl_k reprezintă numărul întrerupătoarelor din camera k care aprind lumini;
- $ap_k[j]$ reprezintă cea de a j-a cameră în care se aprinde lumina din camera k;

- $nr2_k$ reprezintă numărul întrerupătoarelor din camera k care sting lumini;
- $st_k[j]$ reprezintă cea de a j-a cameră în care se stinge lumina din camera k.

Date de ieşire

Rezultatele se vor scrie în fişierul text **VILA.OUT** în următorul format:

dmin
ch_1 cam_1
ch_2 cam_2
...
ch_{dmin} cam_{dmin}

unde:

- dmin reprezintă durata minimă a drumului determinat;
- ch^k este unul din următoarele caractere:
  'a' dacă se aprinde becul în camera cam_k,
  's' dacă se stinge becul în camera cam_k,
  'm' dacă se trece în camera cam_k;
- cam_k reprezintă numărul camerei în care se execută operaţia ch_k.

Numerele situate pe aceeaşi linie sunt separate printr-un singur spaţiu atât în fişierul de intrare cât şi în fişierul de ieşire.

În cazul în care nu există soluţie, în fişierul de ieşire se va scrie: 'Nu exista solutie'.

Exemplu

VILA.IN	VILA.OUT
3	6
2 2 3	a 2
2 1 3	m 2
2 1 2	a 3
1 2	m 3
1 3	s 1
0	s 2
0	
1 3	
2 1 2	

P059921: Perechi

Prof. Marinel Şerban, Liceul de Informatică "Grigore Moisil", Iaşi

Se consideră şirul numerelor naturale prime:

2, 3, 5, 7, 11, 13, 17,...

Din acesta se construieşte un al doilea şir astfel: fiecare număr prim de pe o poziţie impară se concatenează cu numărul prim următor. Se obţine şirul:

23, 57, 1113,...

Apoi, din acest şir se formează un al treilea şir care conţine doar numerele prime:

23, 3137,...

Scrieţi un program care pentru un număr oarecare n ($50 \le n \le 200$), citit de pe prima linie a fişierului text **PERECHI.IN**, furnizează al n-lea număr din acest al treilea şir, număr care va fi scris pe prima linie a fişierului text **PERECHI.OUT**.

Exemple

PERECHI.IN	PERECHI.OUT
1	23

PERECHI.IN	PERECHI.OUT
2	3137

P059922: Legenda

Cătălin Frâncu, student, Universitatea Politehnică, Bucureşti

Printre concurenţii Olimpiadei de Informatică circulă următoarea legendă. În timpul Turnirului Interfeudal de Informatică din anul 1299, infama vrăjitoare Exponentia, orbită de gelozie, l-a blestemat pe viteazul cavaler Algoritmus, a cărui inimă era dăruită prea nobilei Polinomia, cu următorul blestem (exprimat în cuvinte mieroase şi perfide):

"O, preanobile Algoritmus! Iată aici $N \le 100$ *suluri de papirus de diverse lungimi* $S_1, S_2, ..., S_N$. *Lungimea fiecărui papirus este un număr întreg de ţoli, cel puţin unul şi cel mult o sută de ţoli. Pe fiecare ţol al acestor papirusuri este gravat un număr cuprins între* 1 *şi* 100. *Întrucât cunoşti disprețul meu pentru numerele cu soţ, vei şti, neînfricate cavaler, că suma lungimilor papirusurilor este un număr fără soţ. Mai înainte ca Polinomia, pe care o iubeşti, să fie a ta, Algoritmus, îţi cer să lipeşti aceste papirusuri cap la cap într-o asemenea ordine, încât numărul situat la jumătatea papirusului rezultat să fie cât mai mare cu putinţă. Să nu cutezi însă a rupe papirusurile ori a le întoarce cu josul în sus, căci Polinomia va fi pierdută pe veci."*

Sărmanul Algoritmus, neavând altceva de făcut, s-a apucat cu răbdare să încerce fel de fel de modalităţi de a aşeza papirusurile, şi se spune că asta face şi astăzi, cufundându-se pe niveluri din ce în ce mai adânci în mrejele vrăjitoarei Exponentia. Între timp anii au trecut, papirusurile s-au cam ferfeniţit, frumuseţea Polinomiei s-a cam dus şi există zvonuri că s-ar fi măritat. Legenda spune însă că *"Vrăjitoarea poate fi nimicită, iar cavalerul salvat, numai de un program scris în cetatea Mediaşului, care să dezlege misterul* ***în nu mai mult de două secunde****. Autorul acestui program va avea mult de agonisit"*. Nu ştim cât, dar credem legenda pe cuvânt.

Pentru comoditatea voastră, am cules informaţii despre mărimile şi conţinutul papirusurilor în fişierul text **PAPIRUS.IN**, care are formatul:

N
S_1 X(1,1) X(1,2) ... X(1,S_1)
S_2 X(2,1) X(2,2) ... X(2,S_2)
.....
S_N X(N,1) X(N,2) ... X(N,S_N)

În acest fişier, N este numărul de papirusuri, S_i este lungimea celui de-al i-lea papirus, iar X(i,j) este valoarea celui de-al j-lea număr de pe al i-lea papirus.

Răspunsul trebuie să-l dați în fișierul **PAPIRUS.OUT**, indicând numerele papirusurilor în ordinea în care trebuie ele lipite pentru a obține un element cât mai mare în mijlocul papirusului rezultat. Numerele se vor tipări în formatul:

P_1 P_2 ... P_N, unde $1 \le P_i \le N$ $\forall i$ și $P_i \ne P_j$ $\forall i \ne j$

Exemplu

PAPIRUS.IN	PAPIRUS.OUT
4	3 2 1 4
5 4 7 3 1 9	
3 2 5 8	
7 6 4 3 9 5 2 6	
4 4 1 3 8	

Interpretarea este: dacă lipim, în această ordine, papirusurile 3, 2, 1 și 4, obținem papirusul (6, 4, 3, 9, 5, 2, 6, 2, 5, 8, 4, 7, 3, 1, 9, 4, 1, 3, 8). Acesta conține în mijloc elementul 8, și nu putem să aranjăm altcumva papirusurile pentru a obține în mijloc un element mai mare.

Dacă există mai multe soluții, se va tipări una la alegere. Mult succes, prințese și cavaleri!

P059923: Un dreptunghi

Prof. Rodica Pintea, Liceul "Grigore Moisil", București și Iuliu Vasilescu, student, Universitatea Politehnică, București

Se consideră o tablă pe care sunt desenate M×N ($1 \le N$, $M \le 20000$) pătrate. Pe această tablă este plasat, la coordonate necunoscute un dreptunghi de dimensiuni, de asemenea necunoscute. Pentru aflarea poziției și a dimensiunilor dreptunghiului Alinuța aruncă bile de la marginile tablei în lungul liniilor sau coloanelor. Dacă bila izbește dreptunghiul, se întoarce, în caz contrar trece mai departe. Alinuța ne comunică rezultatele aruncărilor, falsificând unele dintre ele. Să se determine poziția și dimensiunile dreptunghiului astfel încât numărul de rezultate falsificate să fie minim. Dacă există mai multe soluții posibile, determinați-o pe cea pentru care aria dreptunghiului este maximă.

Observații:

- bila se poate arunca de mai multe ori pe aceeași linie sau coloană (eventual unele rezultate fiind falsificate);
- dimensiunile dreptunghiului pot fi și 0 (dreptunghiul poate fi un segment sau un punct). Un segment oprește doar bilele care vin perpendicular pe el, iar un punct nu oprește bile.

Date de intrare

Formatul fișierului de intrare **DREPT.IN**:

- pe prima linie sunt scrise două numere naturale M și N, reprezentând numărul de linii, respectiv numărul de coloane ale tablei;
- pe cea de a doua linie este scris un număr natural Q, ($1 \le Q \le 20000$) reprezentând numărul de aruncări pe linii;
- pe următoarele Q linii sunt perechi de numere naturale:

L_1 R_1
L_2 R_2
...
L_Q R_Q

unde linia L_i R_i are semnificația: aruncarea pe linia L_i are rezultatul R_i (valoarea 0 semnifică faptul că bila a trecut, valoarea 1 semnifică faptul că bila s-a întors).

Pe următoarea linie avem un număr natural P, reprezentând numărul de aruncări pe coloane ($1 \le P \le 20000$).

Pe următoarele P linii sunt perechi de numere naturale:

C_1 S_1
C_2 R_2
...
C_P S_P

unde linia C_i S_i are semnificația: aruncarea pe coloana C_i are rezultatul S_i (valoarea 0 semnifică faptul că bila a trecut, valoarea 1 semnifică faptul că bila s-a întors).

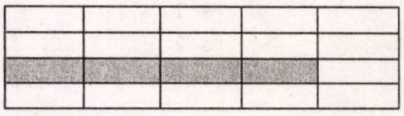
Date de ieșire

Fișierul de ieșire **DREPT.OUT** va conține patru numere naturale: L C H W, unde L, C reprezintă linia și coloana colțului stânga jos al dreptunghiului determinat (pătratul 1, 1 este colțul stânga jos al tablei) iar H, W reprezintă înălțimea și lățimea dreptunghiului.

Exemplu:

DREPT.IN	DREPT.OUT
4 5	2 1 1 4
3	
4 0	
1 0	
5 0	

Figura de mai jos corespunde exemplului:



P059924: Submulțimi

Prof. Ovidiu Domșa, Colegiul "Horea, Cloșca și Crișan", Alba Iulia

Se dă o mulțime cu m elemente numere naturale strict pozitive, a căror sumă este mai mică decât 400000.

Să se determine două submulțimi *disjuncte* ale mulțimii date având proprietatea că *suma elementelor unei mulțimi este egală cu suma elementelor celei de a doua.*

Date de intrare

Datele de intrare se citesc din fișierul de tip text **SUB.IN**. Fișierul are următoarea structură:

Pe prima linie este scris numărul întreg m ($3 \le m \le 30$), reprezentând numărul elementelor mulțimii.

Pe următoarea linie sunt scrise elementele mulțimii, m numere naturale pozitive diferite, separate printr-un spațiu.

Date de ieşire

Rezultatele se vor scrie în fişierul `SUB.OUT`.

- pe prima linie se va afla numărul elementelor unei submulţimi;
- pe a doua linie se vor scrie elementele primei submulţimi separate printr-un spaţiu;
- pe a treia linie se scrie numărul elementelor celei de a doua submulţimi;
- pe a patra linie vor fi scrise elementele celei de a doua submulţimi separate printr-un spaţiu.

Dacă nu există soluţie fişierul va conţine pe prima linie caracterele `NU`.

Exemplu

```
SUB.IN
7
1 10 4 17 124354 25 124330
SUB.OUT
4
1 10 17 124330
2
4 124354
```

Premii I

Clasa IX

Dumitran Daniel, Bucureşti
Pantiş Mihai, Cluj
Dascălu Vlad, Bacău

Clasa X

Pop Ionel, Sălaj
Pătraşcu Mihai, Dolj

Clasa XI

Ştefan Radu, Braşov
Dincă Sergiu, Dâmboviţa
Litvin Andrei, Cluj

Clasa XII

Andoni Alexandru, Rep. Moldova
Vancea Andrei, Cluj
Dumitru Bogdan, Prahova

Lotul naţional de informatică

Proorocu Angel
Vancea Andrei
Dumitru Bogdan
Cârcu Adrian
Alexandrescu Cristian
Gheorghe Andrei
Kuczapski Artur
Costache Octavian
Erzse Gabriel
Manea Ioan
Oprean Daniel
Bagosi István
Ileană Ioana
Ştefan Radu
Batog Bogdan
Pătraşcu Mihai
Leu Ioan Tudor

Premii II

Clasa IX

Pora Lucian, Alba
Margine Cornel, Rep. Moldova
Hirjoc Bogdan, Alba
Crăciun Ciprian, Hunedoara

Clasa X

Alexandru Cătălin, Bucureşti
Gheorghe Andrei, Bucureşti
Andreica Mugurell, Bucureşti
Oprea Alexandru, Mureş
Drumea Adrian, Bucureşti
Stan Bogdan, Prahova
Gheţu Florin, Vrancea

Clasa XI

Ileană Ioana, Alba
Proorocu Angel, Prahova
Oprean Daniel, Alba
Leu Ioan Tudor, Vrancea
Andrei Marius, Bucureşti
Iană Ionel, Olt

Clasa XII

Szász Pál, Bihor
Drula Cătălin, Bucureşti
Cârcu Adrian, Bistriţa-Năsăud
Dumitraşcu Irina, Bucureşti
Stanciu Adrian, Bucureşti
Erzse Gabriel, Bihor
Caloianu Ciprian, Neamţ

Sponsori:

ROMSYS
SOFTWIN
EDITURA ALL
PREFECTURA JUDEŢULUI SIBIU
PRIMĂRIA MUNICIPIULUI MEDIAŞ
ŞCOALA NAŢIONALĂ DE GAZ
ş.a.

Premii III

Clasa IX

Mogoreanu Nicolae, Rep. Moldova
Raţiu Cosmin, Mureş
Toth Cristian, Timiş
Nicolae Bogdan, Sibiu
Benea Andrei, Vrancea

Clasa X

Trifan Ionuţ, Buzău
Crişan Daniel, Vâlcea
Alexandrescu Cristian, Botoşani
Banu Cosmin, Dâmboviţa
Sabău Florin, Caraş-Severin
Bucur Adrian, Cluj
Costescu Andrei, Caraş-Severin
Marina Constantin, Alba
Păun Marulis, Giurgiu
Patcas Csaba, Bihor

Clasa XI

Kuczapski Artur, Bihor
Popa Claudiu, Bucureşti
Clocotici Andrei, Iaşi
Glişea Călin, Bihor
Prisecaru Lucian, Vaslui
Popa Alexandru, Bucureşti
Sauciuc Raluca, Bucureşti
Cibu Cristian, Vâlcea
Dusceac Silvia, Vâlcea

Clasa XII

Batog Bogdan, Bucureşti
Matei Andi, Argeş
Bagosi István, Bihor
Vingărzan Dragoş, Alba
Pietrăroiu Marius, Prahova
Beu Vlad, Sibiu
Cozma Lucian, Vaslui
Costache Octavian, Prahova
Banias Ovidiu, Timiş

REȚELE NEURONALE cu propagarea înapoi a erorii

Lucian Brâncovean

Încă de la începutul cercetărilor în domeniul inteligenței artificiale s-au conturat două direcții principale: modelul logico-simbolic și modelul conexionist. O nouă tendință în acest domeniu este reprezentată de algoritmii genetici și de calculul evolutiv.

Introducere

Modelul logico-simbolic se bazează pe reprezentarea cunoștințelor unui sistem în mod simbolic și pe utilizarea unor metode logice în scopul deducerii altora noi. Exemple bine cunoscute în acest sens sunt programele de șah și sistemele expert.

Modelul conexionist se bazează pe o structură asemănătoare creierului uman alcătuită din elemente interconectate analog neuronilor, creând conceptul de *rețele neuronale artificiale*. În acest caz informația memorată în sistem nu este reprezentată în mod clar, simbolic, ci ea se află distribuită în mod difuz în toate elementele rețelei. Marele avantaj al acestui model este capacitatea sa de a învăța din exemple și de a aplica regulile învățate pe cazuri noi. În mod normal, pentru rezolvarea unei clase de probleme cu ajutorul calculatorului se utilizează o metodă clară, un algoritm alcătuit din etape și decizii bine definite, executate într-o anumită succesiune. Unele clase de probleme necesită însă algoritmi de o asemenea complexitate încât construirea lor se poate dovedi o sarcină deosebit de dificilă, uneori imposibilă. În cazurile în care aceste probleme sunt totuși abordabile prin metode obișnuite, ele cer mari resurse de timp și putere de calcul. Metoda neconvențională a calculului neuronal ar putea face aceste cazuri nu doar abordabile, ci uneori deosebit de simple în raport cu algoritmii complicați existenți și folosiți în mod curent. Etapa de construcție a modelului și a algoritmului corespunzător problemei se reduce la antrenarea rețelei pe o mulțime considerabilă de exemple.

Domeniul rețelelor neuronale este însă un domeniu vast. Primul model de neuron a apărut în 1943 din colaborarea unui matematician cu un neurofiziolog. De atunci, au apărut mai multe moduri de aplicare a paradigmei conexioniste ca: *perceptronul multistrat, mașina Boltzmann, rețele asociative, rețele neuro-fuzzy, propagarea înapoi a erorii* etc. Aceasta din urmă - cunoscută și sub numele de ***regula delta generalizată*** - este în acest moment cea mai importantă și mai utilizată metodă pentru instruirea rețelelor neuronale. De studiul ei ne vom ocupa în continuare.

În general unele categorii specifice de probleme sunt potrivite pentru această abordare.

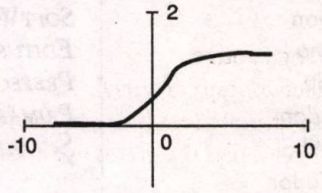
Exemple:

– recunoașterea modelelor în imagini;
– recunoașterea caracterelor;
– recunoașterea vorbirii;
– în general, clasificarea seturilor de date de intrare.

Ideea - asemănarea cu creierul

Ideea inițială a pornit de la asemănarea cu sistemul nervos uman. Acesta este alcătuit dintr-o multitudine de celule specializate, numite ***neuroni***, interconectate, unele având rol de culegere a informației, altele de transmitere a comenzilor către alte mecanisme, iar majoritatea cu scopul de a prelucra și memora informația, într-un mod ce ne rămâne încă puțin cunoscut. Toate aceste celule comunică prin ***impulsuri nervoase*** transmise prin conexiunile lor. Memorarea informației pare a fi realizată prin modificarea conductibilității acestor căi, în sensul creșterii ei în momentul trecerii repetate a unor semnale pe același traseu.

Structura neuronului



Elementul principal al unei rețele este ***neuronul***. El se constituie ca o unitate de prelucrare a informației în formă numerică, având numeroase intrări și o singură ieșire, a cărei valoare este însă transmisă în mai multe direcții, spre alți neuroni. Un neuron este descris de o ***funcție de activare*** care ne indică valoarea ieșirii, ca funcție de o valoare ce reprezintă suma intrărilor sale. Această funcție este aceeași pentru toți neuronii din rețea și poate avea diferite forme. O primă formă des întâlnită este ***funcția prag*** ce are doar două valori posibile, de obicei 0 și 1, după cum argumentul este sau nu, mai mare decât o anumită valoare fixă, numită ***prag***. O altă variantă este ***funcția identică***, ce are ca ieșire chiar valoarea de intrare ($f(x)=x$). Se mai întâlnește și o combinație a acestora de forma

$$f(x)=\begin{cases}0, & x\le 1\\ x, & x>0\wedge x<1\\ 1, & x\ge 1\end{cases}$$

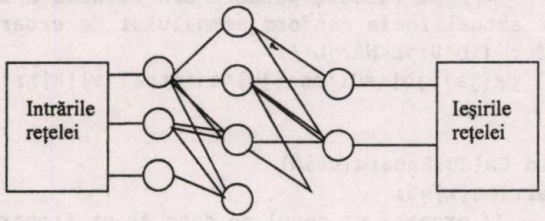
Însă, în cazul de față cea mai productivă se dovedește a fi o funcție de forma $f(x)=\frac{1}{1+e^{-kx}}$.

Straturile și ponderile

Neuronii din rețea sunt grupați în ***straturi***. Orice neuron este conectat la intrare cu ieșirile tuturor neuronilor din stratul precedent, iar ieșirea sa cu intrările tuturor neuronilor din stratul următor. Într-un același strat neuronii nu sunt conectați între ei. În acest mod propagarea informației prin rețea se face unidirecțional, dinspre intrări spre ieșiri.

Există două straturi cu destinație specială: *cel de intrare* — primul în ordinea arătată mai sus — și *cel de ieșire* — ultimul în acest caz. *Numărul neuronilor de intrare* este de obicei numărul de date ale problemei, fiecare primind la intrare unul dintre acești parametri. *Numărul neuronilor de ieșire* este egal cu numărul claselor în care pot fi separate datele de intrare. Reprezentantul clasei în care este încadrat setul de date prezentat la intrare va avea ieșirea 1, iar ceilalți 0.

Conexiunile dintre straturi sunt caracterizate fiecare de o ***pondere***, un număr real cu care se înmulțește ieșirea unui neuron înainte de a fi prezentată la intrarea următorului neuron. Din aceste ponderi constă puterea întregii rețele — prin ajustarea lor se realizează ***memorarea informației***.



Modul de funcționare

Să urmărim acum o etapă de funcționare.

Rețelei i se prezintă un set de date de intrare. Fiecare dintre valori este trimisă la intrarea unuia din neuronii din stratul de intrare.

Aceștia nu fac efectiv nici o prelucrare, valorile sunt trimise mai departe spre următorul strat, prin conexiuni, ajustându-se cu ponderile respective.

Fiecare neuron din stratul ascuns efectuează următorul calcul: se adună toate intrările sale; rezultatului i se aplică funcția de activare a neuronului, funcția sigmoidală în cazul acesta; valoarea rezultată este trimisă mai departe prin ponderi la neuronii stratului următor. Această operațiune se repetă pentru toate straturile rămase, valorile rezultate la ieșirea ultimilor neuroni fiind chiar ieșirile rețelei.

Vă veți întreba, și pe bună dreptate, ce determină acest mecanism să producă rezultatele așteptate. Răspunsul este: *antrenamentul*. Să vedem cum se antrenează o astfel de rețea. La început i se prezintă rețelei un set de date și se calculează ieșirile corespunzătoare. Diferența dintre rezultatul obținut și cel dorit reprezintă eroarea. Această diferență se determină pentru fiecare neuron final și apoi se trimite înapoi prin rețea sub forma unui semnal de eroare. Acesta se calculează cu formula $SE=DS(ACT)*(IR - ID)$, unde IR este ieșirea realizată prin calcul, ID este ieșirea dorită, ACT este activarea acestui neuron în calculul ieșirii, $DS(x)$ este derivata funcției de activare a neuronului. În continuare, semnalul de eroare se propagă înapoi prin rețea în mod asemănător cu informația, dar în sens invers. Pentru un neuron dintr-un strat ascuns SE este suma semnalelor de eroare ale neuronilor din stratul următor, fiecare înmulțită cu ponderea conexiunii prin care se transmite înapoi această valoare. În final, ponderea fiecărei conexiuni se ajustează cu valoarea $AJ=CI*SE*S(ACT)$, unde CI este rata de instruire, SE este semnalul de eroare ACT este activarea și $S(x)$ este funcția de activare a neuronului. În acest fel întreaga rețea va fi modificată pentru a produce un răspuns mai apropiat de cel dorit. Procedeul se repetă pentru toate seturile de date prezente ca exemple pentru problema în cauză. Eroarea apărută la fiecare set este memorată și se calculează o eroare medie. Întregul proces este repetat până când eroarea medie atinge o valoare destul de scăzută.

Implementarea

Problema rezolvată de programul prezentat aici este relativ simplă. Programul trebuie să determine în care parte față de prima bisectoare se găsește un punct. Intrarea este alcătuită din coordonatele carteziene ale punctului, iar ieșirea dintr-o valoare reală — 0 pentru o parte, 1 pentru cealaltă. Datorită modului de funcționare a rețelei ieșirea va avea valori intermediare, apropierea lor de 1 sau 0 fiind și în funcție de eroarea maximă acceptată la antrenarea rețelei. Datele de intrare sunt formate din 6 puncte aflate de o parte și de alta a primei bisectoare, având coordonate întregi din mulțimea {1,2,3}. Problema poate părea prea simplă, dar este potrivită pentru testarea rețelei în condițiile în care interpretarea se face foarte ușor, iar timpii de antrenament sunt reduși. Se poate observa ușor influența parametrului CI (rata de instruire) și a erorii maxime asupra timpului de antrenament. Se poate experimenta cu diferite numere de neuroni în stratul ascuns.

Listing: NEURONI.CPP

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <values.h>
#define NI 2          // nr. neuroni de intrare
#define NA 2          // nr. neuroni ascunsi
#define NF 1 // nr. neuroni de iesire (finali)
#define CI (0.3) // constanta rata de instruire
#define NS 4          // nr. seturi de date
#define EMAX 0.005 // eroarea maxima acceptabila
float pa[NI][NA];  // ponderile intre NI si NA
float pb[NA][NF];  // ponderile intre NA si NF
float intra[NI][NS]= {{1,2,1,2},{1,1,2,2}};
                   // vectorii de intrare
float ies[NF][NS]= {{1,1,0,0}};
                   // vectorii de iesire
float error;       // eroarea medie globala
float h[NA];    // activarile stratului ascuns
float i[NA];    // iesirile  stratului ascuns
float y[NF]; // activarile stratului de iesire
float o[NF];               // iesirile retelei
float sef[NF];
      // semnal de eroare pt stratul de iesire
float sea[NA];
        // semnal de eroare pt stratul ascuns
float errloc[NS];
     // eroarea locala a fiecarui set de date
int k,ja,jb;          // al catelea set de date
unsigned long int tmp;// al catelea antrenament
float s(float in)
{  return (1/(1+exp(-in)));
// functie sigmoidala de activare a neuronului
}
float ds(float in)
// derivata functiei sigmoidale
{ return (exp(in)/((exp(in)+1)*(exp(in)+1)));
}
void InitNet(void)
{ randomize();
  int i,j;
  for(i=0;i<NI;i++)
  for(j=0;j<NA;j++)
  pa[i][j]=( (rand() % 30) + 1.0 )/100;
             // valori mici,diferite de zero
  for(i=0;i<NA;i++)
    for(j=0;j<NF;j++)
      pb[i][j]=( (rand() % 30) + 1.0 )/100;
}
void PrintNet(void)
{ int i,j;
  for(i=0;i<NI;i++)
  { for(j=0;j<NA;j++)
    printf( "%f | ",pa[i][j] ); printf("\n");
  }
  printf("\n");
  for(i=0;i<NA;i++)
  {
  for(j=0;j<NF;j++)
   printf( "%f | ",pb[i][j] ); printf("\n");
  }
  printf("\n");
}
```

```
void ActivNeuroniAscunsi(void)
{ for (ja=0;ja<NA;ja++)
                  // pt fiecare neuron ascuns
  { h[ja]=0;
    for (jb=0;jb<NI;jb++)
                  // se calculeaza activarea
    { h[ja]+=(intra[jb][k])*(pa[jb][ja]);
    }
    i[ja]=s(h[ja]);   // se calcueaza iesirea
  }
}

void ActivNeuroniIesire(void)
{ for (ja=0;ja<NF;ja++)
            // pt fiecare neuron de iesire
  { y[ja]=0;
    for (jb=0;jb<NA;jb++)
                 // se calculeaza activarea
    { y[ja]+=(i[jb])*(pb[jb][ja]);
    }
    o[ja]=s(y[ja]);   // se calcueaza iesirea
  }
}

void SemnalEroareIesire(void)
{ for (ja=0;ja<NF;ja++)// pt fiec. neuron final
  sef[ja] = ds( y[ja] ) * (ies[ja][k] - o[ja]);
}

void SemnalEroareAscuns(void)
{ for (ja=0;ja<NA;ja++)
                  // pt fiecare neuron ascuns
  { sea[ja]=0;
    for (jb=0;jb<NF;jb++)
      sea[ja]=sef[jb]*pb[ja][jb];
    sea[ja]*=ds(h[ja]);
  }
}

void ActualPonderiIesire(void)
{ for (ja=0;ja<NA;ja++)
        // pt fiecare pondere din stratul b
    for (jb=0;jb<NF;jb++)
// se actualizeaza conformsemnalului de eroare
      pb[ja][jb]+=CI*sef[jb]*s(h[ja]);
}

void ActualPonderiAscuns(void)
{ for (ja=0;ja<NI;ja++)
       // pt fiecare pondere din stratul a se
 // actualizeaza conform semnalului de eroare
  for (jb=0;jb<NA;jb++)
    pa[ja][jb]+=CI*sea[jb]*s(intra[ja][k]);
}

void CalculEroare(void)
{ errloc[k]=0;
     // eroarea pt setul de date #k pt fiecare
          // neuron final semisuma patratelor
  for (ja=0;ja<NF;ja++)
  errloc[k]+=pow((ies[ja][k]-o[ja]),2);
  errloc[k]/=2;
  error+=errloc[k];  // eroarea pt acest ciclu
}
```

```
void TrainNet(void)
{
  tmp=0;
  do                    //un ciclu de antrenament
  { error=0;
    for(k=0;k<NS;k++)
              // pentru fiecare vector de date
    {
      ActivNeuroniAscunsi();
           // calculeaza activarea si iesirea
             // stratului de neuroni ascunsi
      ActivNeuroniIesire();
           // calculeaza activarea si iesirea
              // stratului de neuroni finali
      SemnalEroareIesire();
          // se calculeaza semnalul de eroare
      SemnalEroareAscuns();
      ActualPonderiIesire();
             // se actualizeaza ponderile cu
      ActualPonderiAscuns();
                          // semnalul de eroare
     CalculEroare(); // se calculeaza eroarea
    }
   printf("%lu error : %.10f\n",++tmp,error);
  } while (EMAX <= error);
      //pana cand eroarea este destul de mica
}
void TryNet(void)          //utilizarea retelei
{
  printf(" Trying Net ... \n");
  intra[0][0]=2;        // vectorii de intrare
  intra[1][0]=1;      // pentru testele finale
  intra[0][1]=1; intra[1][1]=2;
  intra[0][2]=1; intra[1][2]=1;
  intra[0][3]=2; intra[1][3]=2;
  for (k=0;k<NS;k++)
  {
  ActivNeuroniAscunsi();
           // calculeaza activarea si iesirea
             // stratului de neuroni ascunsi
  ActivNeuroniIesire();
           // calculeaza activarea si iesirea
              // stratului de neuroni finali
  for (ja=0;ja<NF;ja++)
              // pt fiecare neuron de iesire
printf("setul %d iesirea %d=%f \n",k,ja,o[ja]);
  };
}
void main(void)
{
  clrscr();
  InitNet();
  PrintNet();
  getch();
  TrainNet();
  TryNet();
  PrintNet();
  getch();
}
```

Programul folosește o rețea cu doi neuroni de intrare (ascunși), și unul de ieșire. Se afișează ponderile inițiale create aleator, se așteaptă apăsarea unei taste, apoi se începe antrenarea rețelei. Eroarea la fiecare pas este afișată pe ecran. Acest proces poate dura o vreme, în funcție de calculatorul folosit. La sfârșit se afișează ponderile rețelei și se testează rețeaua cu o serie de puncte.

Posibile îmbunătățiri

Programul suportă, evident, numeroase îmbunătățiri. Totuși, el este prezentat în această formă în scopuri didactice.

O primă idee ar fi modificarea numărului de straturi ascunse. Pentru rezolvarea unor probleme dificile este uneori necesar ca rețeaua să conțină două sau chiar mai multe straturi ascunse. Procedeul de actualizare a ponderilor corespunzătoare acestora este analog cu cel folosit pentru unicul strat ascuns din program iar implementarea acestuia nu ridică absolut nici o problemă.

În cazul de față, datorită problemei alese, timpii de antrenament sunt reduși și nu constituie o problemă, la fel seturile de date. Pentru probleme grele și structuri ale rețelei mai complexe — mulți neuroni pe strat, mai multe straturi — timpul ar putea deveni o problemă. În acest caz apare nevoia de a salva/încărca informațiile din rețea (ponderile).

În antrenarea rețelei pot apărea situații de minim local, când eroarea nu este minimul realizabil, dar orice modificare în configurația ponderilor duce la o eroare mai mare. Aici orice modificare a rețelei va face ca modificarea următoare să fie în direcția opusă, revenindu-se la situația inițială. Pentru evitarea acestui gen de situații se poate efectua o modificare a tehnicii de antrenament bazată pe ideea de a ține cont la ajustarea unei ponderi de ajustarea precedentă aplicată aceleiași ponderi. În acest fel se evită cazurile în care rețeaua se "învârte" în jurul aceluiași punct.

Aplicații

Aplicabilitatea rețelelor neuronale este deosebit de largă datorită proprietăților specifice acestora — pot fi aproximator universal și au capacitatea de a învăța din exemple. Domeniile în care ar putea fi folosite sunt diverse, de la inginerie, informatică, medicină, economie, până la domenii umaniste ca sociologie, lingvistică sau psihologie. Aplicații practice, aflate deja în studiu sunt: recunoașterea formelor plane sau chiar tridimensionale, aici apărând o proprietate interesantă — invarianța la rotație; recunoașterea scrisului, a amprentelor digitale și a vorbirii; recunoașterea unor situații patologice în funcționarea inimii sau a creierului uman, prin analizarea semnalelor EKG respectiv EEG; interpretarea automată a imaginilor sonarului și teledetecția; predicția macroeconomică și a piețelor financiare; eliminarea zgomotului și a interferențelor în telecomunicații.

Bibliografie

D. Dumitrescu, Hariton Costin - Rețele neuronale, teorie și aplicații.

Lucian Brâncovean este elev în clasa a XI-a la Colegiul "Gheorghe Lazăr" din Sibiu. Poate fi contactat prin intermediul redacției.

Modelul RELAȚIONAL

Ion Cozac

Ce este modelul relațional? O caracterizare ar putea fi următoarea: modelul relațional este o tehnică de descriere a datelor cu ajutorul tabelelor și o tehnică de manipulare a acestora cu ajutorul unor operatori specifici calculului relațional și/sau algebrei relaționale, concretizați prin utilizarea unor limbaje specifice - cum ar fi QbE (Query by Example) sau SQL (Structured Query Language).

Modelul relațional este un model abstract care are la bază fundamente matematice solide (teoria algebrică a relațiilor). Principiile modelului relațional au fost descrise prin 1969-1970 de către *E. F. Codd*, cercetător la IBM. El a fost primul care a înțeles că rigoarea matematică poate fi folosită pentru a introduce principii solide în domeniul gestiunii bazelor de date - domeniu care era foarte deficitar la acest capitol.

Modelul entități-legături

În articolul precedent am prezentat o arhitectură clasică a unei baze de date. Aceasta este organizată pe trei nivele:
- *nivelul extern* sau *utilizator*;
- *nivelul conceptual*;
- *nivelul intern*.

Pentru a descrie ***nivelul conceptual*** folosim un model abstract numit ***model entități-legături***.

Elementul fundamental al acestui model este ***entitatea***: acest termen generic desemnează un obiect care face parte dintr-o ***clasă (mulțime) de entități***, toate aceste obiecte sunt similare ca structură, dar pot fi deosebite prin proprietăți specifice (pe care le vom numi în continuare ***atribute***). Deoarece definiția este destul de vagă (imprecisă) vom da câte două exemple preluate dintr-o problemă (simplificată) de evidență a facturilor și dintr-o altă problemă de evidență a personalului. Avem următoarele clase de entități:

1. Mulțimea clienților unui magazin

Fiecare client reprezintă o entitate a acestei clase și are următoarele atribute:
- *cod personal;*
- *nume;*
- *localitate* (de domiciliu);
- *adresa.*

Fiecare entitate este identificabilă prin atributul *cod personal* al acesteia.

2. Mulțimea produselor aflate în evidența magazinului

Toate aceste produse pot fi caracterizate prin:
- *cod produs;*
- *denumire;*
- *preț unitar.*

Fiecare produs este o entitate identificabilă prin codul acestuia.

3. Mulțimea departamentelor unei firme

Orice departament se caracterizează prin: *cod*, *denumire*, *șef*.

4. Mulțimea angajaților unei firme

Fiecare angajat se caracterizează prin:
- *număr legitimație;*
- *nume;*
- *departamentul la care lucrează.*

Un alt element care caracterizează acest model este noțiunea de ***legătură***. Legătura (asocierea) între mai multe clase de entități E_1, ..., E_n (nu neapărat distincte) presupune existența unei mulțimi de valori $(e_1, ..., e_n)$, unde e_i este mulțimea valorilor atributelor unei entități din clasa E_i. În practică de cele mai multe ori întâlnim legături între două clase de entități (n=2); acestea se numesc ***legături binare***.

Un exemplu concret preluat din evidența personalului:

```
(101,"Personal",5001,5001,"Petrescu",101)
```

Primele trei valori sunt preluate din clasa de entități `Departamente`, următoarele trei din clasa `Personal`. Se observă ușor că această mulțime de valori este un element al produsului cartezian al celor două mulțimi: `Departamente` și `Personal`. După ce eliminăm valorile care se repetă (această operație este numită ***proiecție***) obținem:

```
(101,"Personal",5001,"Petrescu")
```

Să clasificăm în continuare legăturile binare după cardinalitatea acestora (câte entități din fiecare clasă intră în cadrul legăturii):

- ***legături de tip*** `1:1` (*one-to-one*); observăm o astfel de legătură în cazul unei evidențe a personalului, care indică faptul că un departament este condus de un șef (un departament nu poate fi condus de mai mulți șefi, o persoană nu poate conduce mai multe departamente);
- ***legături de tip*** `1:n` (*one-to-many*); în evidența personalului remarcăm o astfel de legătură între angajații unui departament și departamentul în cauză (la un departament lucrează mai mulți angajați, un angajat nu poate lucra în cadrul mai multor departamente); la evidența facturilor remarcăm o legătură între `Facturi` și `Clienti` (unui client i se pot întocmi mai multe facturi; nu se poate întocmi o aceeași factură pentru mai mulți clienți);
- ***legături de tip*** `m:n` (*many-to-many*); o astfel de legătură există între `Clienti` și `Produse`: un client poate cumpăra mai multe produse, și în același timp mai mulți clienți pot cumpăra un același sortiment de produse.

După ce vor fi prezentate fundamentele modelului relațional, vom prezenta și tehnici de implementare a acestor tipuri de legături.

Domeniu, atribut, relație

Un *domeniu* este o mulțime de valori scalare (atomice - care nu pot fi descompuse) având același tip.

Exemple:
- mulțimea codurilor personale;
- mulțimea numelor de clienți;
- mulțimea numelor de localități etc.

Cu aceste valori se pot efectua mai multe operații:
- cu majoritatea acestor valori se pot face comparații;
- cu unele valori se pot face și alte operații: o cantitate poate fi înmulțită cu un preț (rezultând o valoare), un preț poate fi înmulțit cu o valoare (în cazul unei reașezări) ș.a.m.d.

Un *atribut* al unei clase de entități este o caracteristică (proprietate) care ia valori într-un anumit domeniu.

Exemple:
- atributele clasei de entități `CLIENTI` sunt: *cod client*, *nume client*, *localitate*, *adresa*;
- atributele clasei de entități `PRODUSE` sunt: *cod produs*, *denumire produs*, *preț unitar*.

O *relație* (în sens algebric) este o submulțime a unui produs cartezian $D_1 D_2 ... D_n$, unde D_i sunt domenii de valori (nu neapărat distincte). Aceasta a fost definiția folosită mult timp și pentru noțiunea de relație în domeniul bazelor de date - modelul relațional. Unul dintre cei mai prestigioși autori de lucrări în acest domeniu, C. J. Date (amintit în articolul precedent) propune o nouă definiție pentru noțiunea de ***relație*** (în domeniul bazelor de date): aceasta se compune din două părți:

- ***antetul relației*** este o mulțime de atribute definite fiecare pe câte un domeniu de valori (aceste domenii nu sunt neapărat distincte):
$\{A_1:D_1, A_2:D_2, ..., A_m:D_m\}$;
- ***corpul relației*** este o mulțime de *tuple* (un tuplu este generalizarea noțiunii de *cuplu*); fiecare tuplu este definit ca o mulțimi de valori ale atributelor definite în cadrul antetului:
$\{\{A_1:v_{11}, A_2:v_{12}, ..., A_m:v_{1m}\}, ...,$
$\{A_1:v_{n1}, A_2:v_{n1}, ..., A_m:v_{nm}\}\}$.

Valoarea *m* (numărul de atribute ale relației) reprezintă ***gradul relației***, iar valoarea *n* (numărul de tuple) reprezintă ***cardinalitatea relației***.

Din definiția de mai sus a noțiunii de ***relație*** rezultă următoarele proprietăți:

1. *Atributele nu sunt ordonate* (ordinea lor nu este semnificativă).

Această proprietate rezultă din faptul că antetul este definit ca o mulțime de atribute (într-o mulțime ordinea elementelor nu este semnificativă);

2. *Atributele sunt distincte* (chiar dacă pot exista două atribute definite pe același domeniu), deoarece elementele unei mulțimi sunt distincte;
3. În cadrul corpului relației *tuplele nu sunt ordonate* - corpul relației este definit ca o mulțime de tuple;
4. *Orice atribut are doar valori atomice*;

Această proprietate poate fi enunțată și astfel:

La intersecția dintre o linie și o coloană se află întotdeauna o singură valoare, și niciodată o colecție de valori; rezultă că o relație nu conține grupuri repetitive; spunem în acest caz că relația se află în ***forma întâi normală***;

5. *Nu există tuple duplicate*, deoarece corpul unei relații este o mulțime de tuple.

Dacă o linie din tabela `Clienti` conține valorile:
`(80001, Ionescu, Bucuresti, Str Libertatii)`
toate aceste valori se află într-o anumită relație: ele definesc identitatea unui client, care are codul personal `80001`, are numele `Ionescu`, domiciliază în localitatea `Bucuresti` la adresa `Str Libertatii`.

Prezentăm în continuare o corespondență între *termenii formali* (definiți mai sus) și *termenii informali* (cei folosiți în mod curent în exploatarea bazelor de date relaționale):

Relație	Tabelă
Tuplu	Linie / înregistrare
Cardinalitate	Număr de linii
Atribut	Coloană / Câmp
Grad	Număr de coloane
Domeniu	Mulțime de valori valide

Corespondența nu trebuie considerată ca o echivalență, deoarece există câteva deosebiri de care trebuie să ținem seama (este de altfel una dintre diferențele pe care le sesizăm între teorie și practică).

Relația este o noțiune teoretică. ***Tabela*** este un obiect concret, care are o anumită reprezentare în calculator - sub forma unui tablou bidimensional. Într-o relație ordinea atributelor și a tuplelor nu este semnificativă. Într-o tabelă există o ordonare atât a coloanelor - dată de ordinea acestora la crearea tabelei, cât și a liniilor - dată de ordinea în care acestea au fost introduse, sau de ordinea unei chei care induce o anumită ordonare a liniilor în cadrul tabelei. O relație este formată întotdeauna din tuple distincte. În multe cazuri o tabelă poate avea linii duplicate.

Cheie primară, cheie externă

O ***cheie candidată*** a unei relații R este o mulțime K de atribute cu următoarele proprietăți:
- *identificare unică*: nu există două tuple distincte în R care să aibă aceeași valoare pentru setul de atribute K; cu alte cuvinte, mulțimea K de atribute identifică în mod unic fiecare tuplu al relației R;
- *nereductibilitate*: nu există o submulțime proprie a lui K (distinctă de K) care să aibă proprietatea de identificare unică.

O cheie este ***simplă*** dacă este formată dintr-un singur atribut și se numește ***compusă*** în caz contrar.

O relație poate avea mai multe chei candidate; una dintre acestea se alege pentru a fi folosită în aplicații ca și cheie de identificare a tuplelor. Cheia candidată folosită în acest scop se numește ***cheie primară***; de obicei se folosesc în acest scop acele chei care reprezintă coduri ale înregistrărilor memorate în baza de date. Un motiv important pentru a justifica o astfel de alegere este următorul: valorile de tip *cod* ocupă foarte puțin spațiu, în comparație cu informațiile pe care le identifică, fapt care conduce la o economie importantă de spațiu în cazul creării unui index, precum și la accelerarea regăsirii informațiilor în cazul în care se fac corelații între tabele.

Să analizăm unul din exemplele prezentate:
- tabela `Clienti` are cheia primară `Codcl`, tabela `Produse` are cheia primară `Codpr`, tabela `Facturi` are cheia primară `Nrfact`; în toate aceste cazuri se verifică ușor cele două proprietăți ale cheii primare - atât *identificarea unică* cât și *ireductibilitatea* sunt evidente;
- tabela `Detalii` are cheia primară compusă din atributele `Nrfact` și `Codpr`; *ireductibilitatea* este evidentă și în acest caz, deoarece este posibil să avem mai multe linii cu aceeași valoare pentru `Nrfact` (când un client cumpără mai multe produse cu aceeași factură), și de asemenea este posibil să avem mai multe linii cu aceeași valoare pentru `Codpr` (când mai mulți clienți cumpără același tip de marfă).

În continuare se definește noțiunea de ***cheie externă*** (sau ***cheie străină***) în conexiune cu noțiunea de *cheie candidată*. Să facem precizarea că în majoritatea situațiilor practice apar legături între tabele care se materializează prin coincidența valorilor cheilor primare și externe.

Fie R_2 o relație. O *cheie externă* din R_2 este o mulțime E_k de atribute cu următoarele proprietăți:
- există o tabelă R_1 (care poate sau nu să coincidă cu R_2) care are o cheie candidată P_k;
- fiecare valoare a setului E_k din R_2 coincide cu o valoare a setului P_k din R_1.

Să analizăm din nou unul din exemplele prezentate:
- tabela `Facturi` are o cheie externă `Codcl`, care este cheie primară în tabela `Clienti`;
- tabela `Detalii` are două chei externe: `Nrfact` care este cheie primară în tabela `Facturi`, și `Codpr` care este cheie primară în tabela `Produse`; am remarcat mai sus că cele două chei externe formează împreună cheia primară a tabelei `Detalii`.

Să facem câteva precizări:
- Prin definiție, *fiecare valoare a unei chei externe trebuie să se regăsească printre mulțimea valorilor cheii candidate corespondente*; reciproca nu este obligatorie: de exemplu, putem înregistra informații în tabela `Clienti` despre un potențial client, chiar dacă acesta încă nu cumpără nimic; putem de asemenea înregistra informații despre un produs pe care îl aducem în magazin chiar dacă el nu a fost încă cumpărat de nici un client;
- *O cheie externă este simplă dacă și numai dacă cheia candidată corespondentă este simplă, și este compusă dacă și numai dacă cheia candidată corespondentă este compusă;*
- *Fiecare atribut component al unei chei externe trebuie să fie definit pe același domeniu al componentei corespondente din cheia candidată;*
- *O valoare a unei chei externe reprezintă o referință către un tuplu care conține aceeași valoare pentru cheia candidată corespondentă*; se pune astfel ***problema integrității referinței***: o bază de date nu trebuie să conțină valori invalide pentru chei externe, altfel spus dacă B referă pe A atunci A trebuie să existe.

Ce se întâmplă dacă ștergem din tabela `Clienti` un client pentru care avem câteva linii în tabela `Facturi`? Dacă operația de ștergere se efectuează fără nici un control, după ștergere nu mai este respectată regula integrității referinței. Pentru a preveni asemenea situații, în tabela `Facturi` precizăm la creare:

```
[ON DELETE option] [ON UPDATE option]
```

unde opțiunea poate fi:
- `RESTRICTED` dacă dorim ca operația cerută pentru tabela `Clienti` să fie respinsă;
- `CASCADES` dacă dorim ca operația cerută pentru tabela `Clienti` să provoace operații (actualizări sau ștergeri) "în cascadă" - se actualizează (sau se șterg) toate liniile afectate din tabela `Facturi`; în acest al doilea caz se cere o atenție deosebită, deoarece o ștergere a unei linii din tabela `Facturi` poate afecta linii din tabela `Detalii`.

Cheile externe se folosesc pentru a implementa legăturile dintre entități. Legăturile de tip one-to-one *și* one-to-many *se implementează introducând în una din tabele o cheie externă, care va face legătura cu cheia primară din tabela corespondentă. O legătură de tip* many-to-many *se implementează introducând o tabelă suplimentară care are o cheie primară compusă, fiecare element al cheii primare fiind o cheie externă. Cele două exemple sunt (sperăm) concludente în acest sens.*

Valori NULL

Există unele situații când regula integrității referinței nu poate fi respectată. Să considerăm următorul exemplu foarte simplu: o bază de date pentru evidența personalului este compusă din două tabele:

```
Departamente(Coddep, Dendep, Sefdep)
Angajati(Nrleg, Numeang, Dep)
```

Observăm că tabela `Departamente` are ca și cheie primară atributul `Coddep`, iar cheie externă `Sefdep`, care este cheie primară în tabela `Angajati`. De asemenea tabela `Angajati` are ca și cheie primară atributul `Nrleg`, și cheie externă `Dep`, care este cheie primară în tabela `Departamente`. În momentul creării celor două tabele nu există nici o informație înregistrată. Dacă dorim să introducem informații despre un departament operația de inserare va fi respinsă deoarece nu avem informațiile despre șeful departamentului respectiv. Dacă dorim să introducem informații despre șeful unui departament operația de inserare va fi de asemenea respinsă deoarece nu avem informațiile despre departamentul respectiv.

Această problemă este rezolvată prin introducerea noțiunii de *valoare* `NULL`; semnificația acestei valori este *valoare necunoscută* sau *informație absentă*. În concordanță cu această nouă noțiune, regulile de integritate se completează astfel:

- ***integritatea entității:*** orice atribut al unei chei primare nu poate avea valoarea `NULL`;
- ***integritatea referinței:*** orice valoare a unei chei externe este fie `NULL`, fie coincide cu o valoare a unei chei candidate corespondente.

Problema enunțată mai sus se poate rezolva în felul următor:

Pasul 1:

se introduc informațiile despre departament, fără a se precizeza informația despre șeful departamentului;

Pasul 2:

se introduc informațiile despre șeful departamentului; în acest moment se poate introduce și informația despre departamentul la care acesta lucrează;

Pasul 3:

se actualizează informația absentă din tabela `Departamente`.

Crearea unei baze de date relaționale în SQL

O ***bază de date relațională*** este o bază de date care este văzută de utilizator ca o colecție de relații (tabele) normalizate (aduse cel puțin în forma întâi normală).

Înainte de a prezenta instrucțiunile ***SQL*** (***Structured Query Language*** - Limbaj Structurat de Interogare) pentru definirea unei baze de date să facem câteva precizări:

1. Numele limbajului este impropriu după cum se poate ușor observa. Acest limbaj ne permite atât definirea datelor, cât și manipularea acestora: inserare, modificare, ștergere, consultare (interogare).
2. Orice operație realizată de o instrucțiune SQL operează cu tabele SQL. O tabelă SQL, spre deosebire de relații, poate avea linii duplicate. De asemenea tabelele SQL au o ordine precizată a coloanelor (de la stânga la dreapta), în schimb ordinea liniilor nu este semnificativă. O tabelă SQL se creează cu ajutorul unei comenzi `CREATE TABLE`:

`CREATE TABLE` *Nume-tabela* `(`*Coloana Reprezentare* `[`*Valoare-implicită*`] [`*Restricții*`], ...);`

Reprezentare. Standardul SQL precizează mai multe tipuri de date scalare (detalii privind modul de reprezentare trebuie căutate în documentația SGBD-ului care se folosește în mod concret), unele dintre acestea fiind prezentate în continuare:

`Char(n)` - o coloană de șiruri de maximum `n` caractere;
`Integer` - o coloană de valori întregi pozitive sau negative (de regulă întregi pe 4 octeți);
`Smallint` - o coloană de valori întregi mici (de regulă întregi pe 2 octeți);
`Float(p)` - o coloană de valori reprezentate în virgulă mobilă;
`Decimal(p,q)` - o coloană de valori zecimale, unde `p` este numărul total de poziții pe care se reprezintă numărul, și `q` este numărul de cifre ale părții zecimale (punctul zecimal nu se reprezintă în mod explicit)
`Date` - o coloană de valori de tip dată calendaristică, pentru reprezentarea cărora se folosește un format specific SGBD-ului folosit;
`Time` - o coloană de valori de tip oră exactă.

Valoare implicită. Pentru fiecare coloană se poate preciza o valoare implicită (care trebuie să se încadreze în tipul de date definit la reprezentare), în cazul în care la inserarea unei linii în tabelă nu este precizată o valoare pentru coloana respectivă. Dacă nu a fost precizată o valoare implicită la creare și nici la inserare, atunci coloana respectivă va primi o valoare `NULL`, dar numai dacă o valoare `NULL` este permisă. O valoare implicită se precizează astfel:

`DEFAULT =` *valoare*

Restricții:

Tip de restricție	Observație
NOT NULL	coloana respectivă nu poate primi valori
NULLCHECK (*condiție*)	se precizează o condiție care va fi verificată în momentul inserării unei linii în tabelă, sau în momentul actualizării unei linii:

Exemplu:

```
CHECK (PRETUNIT > 0)
```

restricție de tip cheie candidată:

```
UNIQUE (Lista-de-coloane);
```

restricție de tip cheie primară:

```
PRIMARY KEY (Lista-de-coloane);
```

restricție de tip cheie externă:

```
FOREIGN KEY (Lista-de-coloane)
REFERENCES Nume-tabela
[ON DELETE option] [ON UPDATE option]
```

Comenzile SQL pentru crearea bazei de date cu care am lucrat până acum pot fi următoarele:

```
CREATE TABLE Clienti (Codcl Integer,
  Nume Char(60), Localit Char(60),
  Adresa Char(60), PRIMARY KEY (Codcl));
CREATE TABLE Produse (Codpr Integer,
  Denumire Char(60), Pretunit Integer,
  PRIMARY KEY (Codpr));
CREATE TABLE Facturi (Nrfact Integer,
  Dataf DATE, Codcl Integer,
  PRIMARY KEY (Nrfact), FOREIGN KEY (Codcl)
  REFERENCES Clienti ON DELETE RESTRICTED
  ON UPDATE RESTRICTED);
CREATE TABLE Detalii (Nrfact Integer,
  Codpr Integer, Cant Integer,
  PRIMARY KEY (Nrfact, Codpr),
  FOREIGN KEY (Nrfact) REFERENCES Facturi,
  FOREIGN KEY (Codpr) REFERENCES Produse;
```

După crearea unei tabele, putem crea și un index pentru aceasta:

```
CREATE INDEX Nume-index ON Nume-tabela;
             (Lista-coloane)
```

Structura unei tabele poate fi modificată ulterior cu ajutorul unei comenzi ALTER TABLE:

```
ALTER TABLE Nume-tabela ADD/MODIFY/DROP;
Nume-coloana Specificatii ...
```

Este posibil să adăugăm o coloană nouă în tabelă, să ștergem o coloană existentă din tabelă, sau să modificăm specificațiile (de tip, de restricții etc.) ale unei coloane existente.

Încărcarea și actualizarea datelor

Introducerea datelor

Instrucțiunea Insert are următoarea sintaxă:

```
INSERT INTO Nume-tabela [(Lista-coloane)]
VALUES (Lista-valori);
```

Datele pot fi introduse în mai multe moduri:

- se precizează lista coloanelor care vor primi valori, și în acest caz celelalte coloane vor primi fie valoarea NULL, fie valoarea implicită specificată la crearea tabelei; lista valorilor corespunde listei coloanelor;

Exemple:

```
INSERT INTO Departamente (Coddep, Dendep)
VALUES (101, 'Personal');
```

- nu se precizează lista coloanelor, caz în care trebuie specificate valorile în listă în conformitate cu lista coloanelor precizată la crearea tabelei; lista valorilor trebuie să fie completă (eventual cu argumente absente precizate explicit);

```
INSERT INTO Departamente
VALUES (101, 'Personal',);
INSERT INTO Angajati
VALUES (5001, 'Petrescu',101);
```

Actualizarea datelor

Instrucțiunea Update are următoarea sintaxă:

```
UPDATE Nume-tabela
SET Coloana = Valoare
[WHERE Conditie];
```

Sunt actualizate liniile din tabelă pentru care este verificată condiția specificată.

Atenție:
Dacă nu s-a specificat nici o condiție sunt actualizate toate liniile tabelei.

Exemplu:

```
UPDATE Departamente
SET Sefdep = 10
WHERE Coddep=100;
```

Ștergerea datelor

Instrucțiunea DELETE are sintaxa următoare:

```
DELETE FROM Nume-tabela
[WHERE Condiție];
```

Sunt șterse liniile din tabelă pentru care este verificată condiția specificată.

Atenție: dacă nu s-a specificat nici o condiție sunt șterse toate liniile tabelei.

Exemplu:

```
DELETE FROM Departamente
WHERE Departament='Personal';
```

Ion Cozac este lector la Universitatea "Petru Maior" din Târgu-Mureș și poate fi contactat prin e-mail la adresa: cozac@uttgm.ro

Informațiile din memoria CMOS

Ion Gelu Tomșa

Memoria de tip CMOS (Complementary Metal Oxide Semiconductor) cu capacitatea de 64 sau 128 octeți este de fapt o memorie de tip RAM, susținută de o baterie. La început această memorie era folosită ca o extensie de tip calendar-ceas pentru sistemele IBM-XT, iar la introducerea seriei PC-AT (Advanced Technology), în 1985, memoria CMOS a devenit o parte integrată a plăcii de bază. Întrucât ceasul și calendarul folosesc împreună numai 14 octeți RAM, restul octeților sunt disponibili pentru păstrarea principalelor informații de configurare a sistemului. Datorită existenței bateriei, conținutul acestei memorii se păstrează și după scoaterea de sub tensiune a calculatorului, permițând actualizarea ceasului-calendar și menținerea celorlalte informații.

Accesarea memoriei CMOS

Memoria CMOS există în afara spațiului normal de adrese și nu poate conține cod direct executabil. Conținutul ei poate fi accesat prin comenzi `IN` și `OUT` trimise la porturile `70h` (112) și `71h` (113). Pentru a citi un octet din memoria CMOS, se trimite o comandă `OUT` portului `70h` cu adresa octetului pe care dorim să-l obținem. Printr-o comandă `IN` la portul `71h` se obține valoarea octetului cerut. Scrierea unui octet se face similar, printr-o comandă `OUT` la portul `70h` cu adresa octetului pe care dorim să-l scriem, urmată de o comandă `OUT` la portul `71h` cu valoarea octetului.

Spațiul de memorie CMOS are o limită superioară de adresare de 128 octeți determinată de structura portului `70h`: numai biții 0-6 sunt folosiți pentru adresare, iar bitul 7 este folosit pentru a activa (`0`) sau a dezactiva (`1`) întreruperile nemascabile (*NMI*). Din această cauză la scrierea/citirea octeților din această memorie adresa se obține efectuând operația:

`80h OR <adresa>.`

Citirea unui octet din memoria CMOS se face astfel:

a) în limbaj de asamblare:

```
adr_octet db ? ; adresa din memoria CMOS
val_octet  db ? ; valoarea octetului
mov  AL, [adr_octet]
out  AL, 70h
in   AL, 71h
mov  [val_octet], AL
```

b) în limbajul Turbo Pascal folosim tabloul predefinit `port` care permite accesul direct, la nivel de octet, la porturile calculatorului:

```
function citire_CMOS(adr_octet:Byte):Byte;
begin
  port[$70]:=adr_octet;
  citire_CMOS:=port[$71]
end;
```

c) în limbajul C apelăm funcțiile:

```
typedef unsigned char Byte;
                /*  definim tipul Byte 0...255 */
Byte  citire_CMOS(Byte adr_octet)
{
  outportb(0x70, adr_octet);
  return(inportb(0x71));
}
```

Scrierea unui octet în memoria CMOS:

a) în limbaj de asamblare:

```
mov AL, [adr_octet]
out AL, 70h
mov AL, [val_octet]
out AL, 71h
```

b) în limbajul Turbo Pascal:

```
procedure scrie_CMOS(adr_octet,val_octet:Byte);
begin
  port[$70]:=adr_octet;
  port[$71]:=val_octet
end;
```

în limbajul C:

```
void scrie_CMOS(Byte adr_octet,Byte val_octet)
{
  outportb(0x70, adr_octet);
  outportb(0x71, val_octet);
}
```

Observații:

1) Forma generală a instrucțiunilor IN, OUT (operațiile cu porturile în limbaj de asamblare) este:

```
IN  dest, port
OUT dest, port
```

în care dest este registrul AL sau AX, iar port este fie o constantă cuprinsă între 0 și 255, fie registrul DX. Semnificația instrucțiunilor este *citirea*, respectiv *scrierea*, de la portul de intrare/ieșire indicat, a unei valori pe un octet (8 biți) sau un cuvânt (16 biți). La procesoarele mai noi (începând cu 80286) regiștrii AL sau AX pot fi înlocuiți cu orice regiștri generali.

2) Funcțiile pentru lucrul cu porturile hardware în limbajul C au prototipurile în fișierul header **dos.h**:

```
unsigned char inportb(int port);
void outportob(int port,unsigned char valoare);
```

3) Dacă memoria CMOS are disponibili numai 64 de octeți, adresele între 40h-7Fh vor oglindi exact conținutul din spațiul de adrese 00h-3Fh.

Organizarea memoriei CMOS

Ceasul sistem

Adresele între 00h-0Eh sunt folosite pentru ceasul de timp real (*RTC - Real Time Clock*) și calendarul sistem și au aceeași semnificație pentru toți producătorii de Bios. Primii 14 octeți sunt folosiți pentru funcțiile cipului de ceas și constau din zece regiștri de scriere/citire date și patru regiștri de stare, dintre care doi regiștri permit operații de scriere/citire, iar ceilalți doi permit numai operații de citire.

Datele sunt reprezentate, unde nu este specificat altfel, pe un octet. Datele reprezentate pe biți folosesc convenția de notare a biților:

0000 0000
7654 3210

cu bitul 7 cel mai semnificativ bit.

Formatul celor zece regiștri de date (octeții 00h-09h) ai ceasului este următorul:

CMOS 00h - RTC - SECUNDE
Secunde ceas: zecimal 00-59, hexa 00-3B.

CMOS 01h - RTC - SECUNDE ALARMĂ
Secunde alarmă: zecimal 00-59, hexa 00-3B; nu are semnificație pentru valori între C0-FF

CMOS 02h - RTC - MINUTE
Minute ceas: zecimal 00-59, hexa 00-3B; nu are semnificație pentru valori între C0-FF

CMOS 03h - RTC - MINUTE ALARMĂ
Minute alarmă: zecimal 00-59, hexa 00-3B; nu are semnificație pentru valori între C0-FFh

CMOS 04h - RTC - ORA
Ore ceas: zecimal 00-23, hexa 00-17 dacă este modul cu 24 ore
- zecimal 01-12, hexa 01-0C dacă este modul cu 12 ore am
- zecimal 81-92, hexa 81-8C dacă este modul cu 12 ore pm

CMOS 05h - RTC - ORA ALARMĂ
Ore alarmă: la fel ca și la ore ceas

CMOS 06h - RTC - ZIUA DIN SĂPTĂMÂNĂ
Ziua din săptămână: 01-07 Duminică=1

CMOS 07h - RTC - DATA
Data din lună: zecimal 01-31, hexa 01-1F

CMOS 08h - RTC - LUNA
Luna: zecimal 01-12, hexa 01-0C

CMOS 09h - RTC - ANUL
Ultimele două cifre ale anului 00-99, hexa 00-63 (vezi și CMOS 32h)

Observație

Valorile sunt reprezentate în binar sau sunt codate BCD (cifre zecimale) iar selecția binar/BCD se face prin bitul 2 al registrului de stare B (0Bh);
- selecția formatului cu 12/24 ore depinde de bitul 1 al registrului B (0Bh);
- alarma se activează când toți cei trei regiștri folosiți de alarmă au valori egale cu cele ale regiștrilor de ceas corespunzători; alarma este accesibilă numai dacă bitul 5 al registrului B (0Bh) are valoarea 1.

Formatul regiștrilor de stare (octeții 0Ah-0Dh) este:

CMOS 0A - RTC - REGISTRUL DE STARE A
(citire/scriere)

Descrierea biților pentru registrul de stare A este:

Bit	Valoare	Descriere
7	1	ciclu de actualizare a ceasului în execuție, ieșirile de date nedefinite
	0	se poate face citirea (bitul 7 este ***Read only***)
6-4		divizor cu 22 stadii
	010	32768 Hz bază de timp (implicit)
3-0		biții pentru selectarea frecvenței de apel a întreruperii
	0000	nici un apel
	0011	122 microsecunde (minim)
	1111	500 milisecunde
	0110	976.562 microsecunde (implicit 1024 Hz)

CMOS 0B - RTC - REGISTRUL DE STARE B
(citire/scriere)
Descrierea biților pentru registrul de stare B este:

Bit	Valoare	Descriere
7	1	permite ciclul de actualizare a ceasului
6	1	permite întreruperea periodică
5	1	permite înreruperea de alarmă
4	1	permite întreruperea cu actualizare la sfârșit
3	1	permite ieșirea square wave
2	1	modul de reprezentare a datelor 0: zecimal (BCD), 1: binar
1	1	selectare mod 24/12 ore: 1 selectează modul 24 ore
0		daylight savings enable

CMOS 0Ch - RTC - REGISTRUL DE STARE C
(numai citire)
Descrierea biților pentru registrul de stare C:

Bit(s)	Descriere
7	Flag de cerere întrerupere, este egal cu 1 când oricare din biții 6-4 ai registrului B sunt setați cu valoarea 1. Generează IRQ 8 when triggered.
6	Flag întrerupere periodică
5	Flag întrerupere de alarmă
4	Flag întrerupere cu actualizare la sfârșit
3-0	Nefolosiți

CMOS 0Dh - RTC - REGISTRUL DE STARE D
(numai citire)
Descrierea biților pentru registrul de stare D:

Bit	Descriere
7	Starea bateriei: 1 indică bateria în stare bună, 0 baterie descărcată sau deconectată
6-0	nefolosiți

Zona din memoria CMOS în afara ceasului sistem

Următorii doi octeți, cu adresele 0Eh și 0Fh, nu sunt specificați pentru sistemele *PC-AT* dar pot avea pentru unele sisteme semnificația:

CMOS 0Eh - IBM PS/2 - octet de stare diagnostic
Descrierea biților pentru octetul de stare în urma diagnosticului pentru sisteme IBM PS/2:

Bit	Val.	Descriere
7	1	indică întreruperea alimentării ceasului
6	1	sumă de control incorectă
5	1	eroare la lista de echipament: rutina POST nu a detectat toate echipamentele din listă
4	1	eroare la dimensiunea memoriei din cauză că rutina POST a determinat altă dimensiune a memoriei instalate
3	1	inițializare eșuată a controlerului sau unității de disc (harddisk defect); nu se poate încărca sistemul de operare de pe harddisk
2	1	oră invalidă
1	1	adaptorul instalat nu corespunde cu descrierea configurației
0	1	time-out în timpul citirii ID adaptorului

CMOS 0Fh - IBM - cod de reset
(IBM PS/2 "Octet de stare la oprire")

Începând cu octetul 10h apare o descriere a periferice-lor și informații de configurare: tipul unităților de floppy disk, hard disk, adaptor grafic, memorie instalată în sistem (de bază și extinsă), parola pentru accesul la informațiile de configurare (BIOS) sau accesul în sistem la pornire, parametrii memoriei cache, configurarea plăcii de bază, sloturi de extensie, parametrii pentru Power Management System (dacă sistemul permite economisirea energiei), asignarea întreruperilor hard, ș.a. Această zonă a memoriei CMOS nu este standardizată, producători diferiți dând interpretări diferite aceluiași octet.

În continuare vom prezenta câteva din informațiile despre componentele instalate în sistem:

CMOS 10h - IBM - tip de unitate floppy
Descrierea biților pentru tipul unităților floppy A/B:

Bit	Descriere
7-4	tipul primei unități floppy (A:)
3-0	tipul celei de-a doua unități floppy (B:)

Valorile posibile pentru tipul unității floppy:

00h	fără nici o unitate
01h	Unitate de 360 KB 5.25" (depășită)
02h	Unitate de 1.2 MB 5.25"
03h	Unitate de 720 KB 3.5"
04h	Unitate de 1.44 MB 3.5"
05h	Unitate de 2.88 MB 3.5"
06h-0Fh	nefolosiți???

Observație

Pentru un calculator cu o singură unitate de 1.44MB acest octet are valoarea 0400h.

CMOS 11h-IBM PS/2 - tipul primului disk fix (00-FFh)
CMOS 12h-IBM PS/2 - tipul celui de al doilea fix (00-FFh)
CMOS 13h- rezervat
CMOS 14h-IBM - octet descriere echipament

Bit	Descriere
7-6	număr unități floppy - 1 00b - 1 unitate 01b - 2 unități 10b - 3 unități 11b - 4 unități

5-4	tip monitor 00b diferit de CGA sau MDA (folosit pentru EGA&VGA) 01b 40x25 CGA, EGA sau VGA 10b 80x25 CGA, EGA sau VGA 11b MDA (monocrom)
3	monitor instalat
2	tastatură instalată
1	coprocesor matematic instalat
0	unitate floppy instalată

CMOS 15h-IBM - memoria de bază în kb (octetul mai puțin semnificativ- low Byte)

CMOS 16h-IBM - memoria de bază în kb (octetul semnificativ - high Byte)

Observație

Valoarea octeților 15h-16h trebuie să fie aceeași cu valoarea păstrată în zona de date BIOS 0:413h și aceeași cu valoarea întoarsă de INT 12h. Pentru un PC având 640k (280h) de memorie convențională va apare valoarea 80h în octetul 15h și valoarea 02h în octetul 16h.

CMOS 17h-IBM - memoria extinsă în KB (octetul mai puțin semnificativ - low Byte)

CMOS 18h-IBM - memoria extinsă în KB (octetul semnificativ - high Byte)

Dimensiunea în KB a memoriei extinse (peste 1Mb) instalate.

CMOS 1Bh-2DH rezervați

CMOS 2Eh-IBM - Standard CMOS Checksum, High Byte

CMOS 2Fh-IBM - Standard CMOS Checksum, Low Byte

Observație

Octeții 2Eh și 2Fh sunt definiți pentru calculatorul original IBM PC-AT și se calculează numai pentru valorile locațiilor 10h-2Dh, valorile din domeniile de adrese 00-0Fh și 30h-3Fh nu sunt incluse.

CMOS 30h - IBM - memoria extinsă în KB (low Byte)

CMOS 31h - IBM - memoria extinsă în KB (high Byte)

Aceste valori sunt aceleași cu cele din octeții 17h-18h.

CMOS 32h - IBM - secolul curent

Acest octet reprezintă valoarea în format BCD pentru secol - în prezent 19.

Actualizarea acestui octet determină compatibilitatea cu trecerea la anul 2000 a BIOS-ului.

CMOS 33h-37h - octeți rezervați de producătorul CMOS: opțiuni pentru Setup, parola pentru AMI Bios - 38h-3Dh etc.

CMOS 38h-3Fh - IBM-PS/2 - parola de activare Setup.

În aceste locații nu veți afla caracterele parolei ci se păstrează valoarea obținută prin aplicarea unei funcții de dispersie (hash) octeților din componența parolei; este posibil ca două (sau mai multe) parole diferite să fie acceptate ca fiind valide dacă au aceeași imagine prin funcția de dispersie. Scrierea de zerouri în octeții rezervați pentru parolă duce la anularea parolei.

În unit-ul **CMOS.PAS** sunt scrise câteva proceduri/funcții folosite în programul **TEST.PAS** care afișează informații despre configurația sistemului. Programul poate fi compilat-rulat și din mediul Borland Pascal pentru Windows dacă se folosește unit-ul wincrt în locul unitului crt.

Listing: CMOS.PAS

```
unit CMOS;

interface
uses crt;
type timp=record
            ora,min,sec:Byte
          end;
     data=record
            secol,an,luna,data,ziua:Byte
           end;
    floppy=(none, D525_360, D525_12,
            D525_720, D35_144, D35_288);
var fdd_A, fdd_B:Byte;

procedure GetTime(var T:timp);
           { citeste ora din ceasul sistem }
procedure SetTime(var T:timp);
           { schimba ora din ceasul sistem }
procedure ScrieOra; { afiseaza ora pe ecran }
procedure GetData(var D:data);
          { citeste data din ceasul sistem }
procedure SetData(D:data);
          { schimba data din ceasul sistem }
procedure ScrieData;
                  { afiseaza data pe ecran }
procedure GetFddType(var fdd_A,fdd_B:Byte);
     { citeste tipul unitatilor de discheta }
function NrFdd:Byte;
      { numarul unitatilor floppy instalate }
function BaseMem:Word;
            { dimensiunea memoriei de baza }
function ExtMem:Word;
            { dimensiunea memoriei extinse }
implementation
procedure BCD2bin(var b:Byte);
 { conversia din format BCD in format binar }
begin
  b:=(b shr 4)*10+(b and $0f)
end;

function bin2BCD(var b:Byte):Byte;
{ conversia din formatul binar in format BCD }
begin
  bin2BCD:=(b div 10) shl 4+(b mod 10)
end;
```

```
procedure GetData(var D:data);
var b:Byte;
begin
  port[$70]:=$32;                { RTC-secolul }
  D.secol:=port[$71];
  BCD2bin(D.secol);
  port[$70]:=9;{ RTC-anul (ultimele 2 cifre) }
  D.an:=port[$71];
  port[$70]:=8;                  { RTC-luna }
  D.luna:=port[$71];
  port[$70]:=7;          { RTC-ziua din luna }
  D.data:=port[$71];
  port[$70]:=6;     { RTC-ziua din saptamana }
  D.ziua:=port[$71];
  port[$70]:=$0B;{ RTC-registrul de stare B }
  b:=port[$71];
  if (b and 4)=0
  then                           { format BCD }
  begin
    BCD2bin(D.an); BCD2bin(D.luna);
    BCD2bin(D.data); BCD2bin(D.ziua)
  end
end;

procedure SetData(D:data);
var b:Byte;
begin
  port[$70]:=$0B;{ RTC-registrul de stare B }
  b:=port[$71];
  port[$70]:=$0B;                { format BCD }
  port[$71]:=b or 4;
  bin2BCD(D.secol);
  bin2BCD(D.an); bin2BCD(D.luna);
  bin2BCD(D.data); bin2BCD(D.ziua);
  port[$70]:=$32;                { RTC-secolul }
  port[$71]:=D.secol;
  port[$70]:=9;{ RTC-anul (ultimele 2 cifre) }
  port[$71]:=D.an;
  port[$70]:=8;                  { RTC-luna }
  port[$71]:=D.luna;
  port[$70]:=7;          { RTC-ziua din luna }
  port[$71]:=D.data;
  port[$70]:=6;  { RTC - ziua din saptamana }
  port[$71]:=D.ziua
end;

procedure GetTime(var T:timp);
var b:Byte;
begin
  port[$70]:=0;                  { RTC-secunde }
  T.sec:=port[$71];
  port[$70]:=2;                  { RTC-minute }
  T.min:=port[$71];
  port[$70]:=4;  { RTC - ore }
  T.ora:=port[$71];
  port[$70]:=$0B;{ RTC-registrul de stare B }
  b:=port[$71];
  if (b and 2)=0
  then { modul am/pm }
    if T.ora>12 then Dec (T.ora, $80+12);
  if (b and 4)=0
  then                           { format BCD }
  begin
    BCD2bin(T.ora);
    BCD2bin(T.min);
    BCD2bin(T.sec)
  end
end;

procedure SetTime(var T:timp);
var b:Byte;
begin
  port[$70]:=$0B;{ RTC-registrul de stare B }
  b:=port[$71];
  port[$70]:=$0B; port[$71]:=(b or 4);
  bin2BCD(T.ora);
  bin2BCD(T.min);
  bin2BCD(T.sec);
  port[$70]:=0;                  { RTC-secunde }
  port[$71]:=T.sec;
  port[$70]:=2;                  { RTC-minute }
  port[$71]:=T.min;
  port[$70]:=4;                  { RTC-ore }
  port[$71]:=T.ora
end;

procedure GetFddType(var fdd_A,fdd_B:Byte);
var Tip:Byte;
begin
  port[$70]:=$10;    { RTC-Floppy drive type }
  Tip:=port[$71];
  fdd_A:=Tip shr 4; fdd_B:=Tip and $0f
end;

function NrFdd:Byte;
var lista:Byte;
begin
  port[$70]:=$14;
  lista:=port[$71];
  if (lista and 1)=1 then NrFdd:=lista shr 6+1
                     else NrFdd:=0
end;

function BaseMem:Word;
var l,h:Byte;
begin
  port[$70]:=$15;    { RTC-Base memory (low) }
  l:=port[$71];
  port[$70]:=$16;  { RTC-Base memory (high) }
  h:=port[$71];
  BaseMem:=h shl 8+l
end;
```

```
function ExtMem:Word;
var l,h:Byte;
begin
  port[$70]:=$17;{ RTC-Extended memory (low) }
  l:=port[$71];
  port[$70]:=$18;{ RTC-Extended memory (high) }
  h:=port[$71];
  ExtMem:=h shl 8+l
end;

procedure ScrieData; { afiseaza data pe ecran }
var d:data;
begin
  GetData(d);
  Write('Astazi este');
  Write(D.data:2,'/',D.luna:2,'/',D.secol);
  if D.an<10 then Write('0');
  Writeln(D.an); Writeln
end;

procedure ScrieOra; { afiseaza ora pe ecran }
var t:timp;
begin
  GetTime(t);
  Write('Este ora: ');
  if t.ora<10 then Write('0',t.ora,':')
              else Write(t.ora,':');
  if t.min<10 then Write('0',t.min,':')
              else Write(t.min,':');
  if t.sec<10 then Write('0',t.sec)
              else Write(t.sec);
  Writeln
end;
End.
```

Listing: TEST.PAS

```
program test;
uses crt, cmos;
procedure TipFdd;
  procedure Tip(f:Byte);
  begin
    case floppy(f) of
      none     : Write('fără');
      D525_360: Write('5.25"-360Kb');
      D525_12 : Write('5.25"-1.2Mb');
      D525_720: Write('5.25"-720Kb');
      D35_144 : Write('3.5" - 1.44Mb');
      D35_288 : Write('3.5" - 2.88Mb')
    end
  end;
begin
  GetFddType(fdd_A,fdd_B);
  case NrFdd of
  1: begin Write('Unitatea A: ');
           tip(fdd_A); Writeln
     end;
  2: begin
       Write('Unitatea A: ');
       tip(fdd_A); Writeln;
       Write('Unitatea B: ');
       tip(fdd_B); Writeln
     end
  end
end;

procedure ScrieMemBaza;
begin
  Writeln('Memorie de baza: ',BaseMem,' Kb');
  Write('Memorie extinsa: ',ExtMem,' Kb')
end;

Begin
  Clrscr;
  Write('Astazi este: '); ScrieData;
  Write(' ora: '); ScrieOra;
  TipFdd;
  ScrieMemBaza;
  Readkey
End.
```

Interfața Bios pentru ceasul de timp real

Operațiile de citire/actualizare a zonei ceas/calendar sunt efectuate prin intermediul întreruperii BIOS 1ah care oferă accesul la ceasul de timp real (*Real Time Clock*) din memoria CMOS și la ceasul intern al sistemului. Acesta este actualizat de aproximativ 18.2 ori pe secundă (adică la un interval de 55ms). Funcțiile oferite de întreruperea `1ah` sunt următoarele:

1) `AH=00h` **Citirea ceasului sistem**

După apel:

- regiștrii `CX` și `DX` conțin numărul de unități de timp de la pornirea calculatorului, `CX` octetul `high` și `DX` octetul `low`
- registrul `AL` are valoarea 0 dacă nu au trecut 24 ore de la ultima punere la 0 a ceasului.

Observație

Ceasul se actualizează de `1193180/65536` (aproximativ 18,2) ori pe secundă. Aceasta înseamnă:

- 18 actualizări pe secundă
- 1092 actualizări pe minut
- 65543 actualizări pe oră
- 1573040 actualizări pe zi.

2) `AH=01h` **Scrierea ceasului sistem**

La apelul întreruperii regiștrii `CX` și `DX` conțin numărul de unități de timp de la pornirea calculatorului, `CX` octetul `high` și `DX` octetul `low`.

3) `AH=02h` **Citirea orei din ceasul de timp real - CMOS**

După apel, în registrul `CH` se obține ora, în registrul `CL` minutul, iar în registrul `DH` secunda. Toate valorile sunt reprezentate în format BCD.

4) AH=03h **Scrierea orei în ceasul de timp real - CMOS**

La apelul întreruperii registrul CH trebuie să conțină ora, registrul CL minutul și registrul DH secunda, toate în format BCD.

5) AH=04h **Citirea datei din memoria CMOS**

După apel în registrul CH se obține secolul, în registrul CL anul, în registrul DH luna și în registrul DL data, toate valorile reprezentate în format BCD.

6) AH=05h **Scrierea datei în memoria CMOS**

La apel registrul CH conține secolul, CL conține anul, DH conține luna și DL conține data în format BCD.

7) AH=06h **Stabilirea orei pentru alarma ceasului de timp real**

Ceasul de timp real oferă posibilitatea activării unei întreruperi de tipul 4ah la o oră fixată prin apelul acestei funcții (întrerupere de alarmă). La apelul acestei funcții registrul CH trebuie să conțină ora, registrul CL minutul, registrul DH secunda, toate în reprezentare BCD. Întreruperea 4ah este activată când apare egalitatea între valorile corespunzătoare ceasului sistem și alarmei din memoria CMOS. Implicit rutina de tratare a acestei întreruperi conține doar o instrucțiune de revenire din întrerupere (IRET), dar poate fi înlocuită cu o rutină scrisă de utilizator. Această întrerupere este disponibilă numai pentru sisteme cu microprocesor 80286 sau mai bun.

8) AH=07h **Anularea ultimei alarme stabilite**

Această funcție permite anularea unei alarme înainte de stabilirea altei alarme.

Observație

La un moment dat poate fi stabilită o singură alarmă. Pentru a schimba ora alarmei trebuie anulată eventual ultima alarmă.

Aceste funcții sunt implementate în unit-ul **RTC.PAS** prezentat mai jos.

Listing: Unit-ul RTC

```
{$F+}
unit rtc;
interface
uses dos;
type data=record
            secol,an,luna,data:Byte
          end;
     timp=record
            ora,min,sec:Byte
          end;

function GetTicks:LongInt;
procedure SetTicks(t:LongInt);
procedure ReadTime(var t:timp);
procedure SetTime(t:timp);
procedure ReadDate(var d:data);
procedure SetDate(d:data);
procedure SetAlarm(t:timp);
procedure ResetAlarm;
implementation
var r:registers;

function BCD2bin(b:Byte):Byte;
                        { conversia BCD binar }
begin
  BCD2bin:=(b shr 4)*10+(b and $0F)
end;

function bin2BCD(var b:Byte):Byte;
{ conversia din formatul binar in format BCD }
begin
  bin2BCD:=(b div 10) shl 4+(b mod 10)
end;

function GetTicks:LongInt;      { citeste nr. }
 { de unitati de timp de la ultima resetare }
begin
  r.AH:=0;
  intr($1a,r);
  GetTicks:=r.CX shl 16 + r.DX
end;

procedure SetTicks(t:LongInt); { seteaza nr. }
 { de unitati de timp de la ultima resetare }
begin
  r.AH:=01;
  r.CX:=t shr 16;
  r.DX:=(t shl 16) shr 16;
  intr($1a,r)
end;

procedure ReadTime;                { citeste ora }
begin
  r.AH:=02;
  intr($1a,r);
  t.ora:=BCD2bin(r.CH);
  t.min:=BCD2bin(r.CL);
  t.sec:=BCD2bin(r.DH)
end;

procedure SetTime;                 { seteaza ora }
begin
  r.AH:=03;
  r.CH:=bin2BCD(t.ora);
  r.CL:=bin2BCD(t.min);
  r.DH:=bin2BCD(t.sec);
  intr($1a,r)
end;

procedure ReadDate;                { citeste data }
begin
  r.AH:=04; intr($1a,r);
  d.secol:=BCD2bin(r.CH); d.an:=BCD2bin(r.CL);
  d.luna:=BCD2bin(r.DH); d.data:=BCD2bin(r.dl)
end;
```

```
procedure SetDate;                { seteaza data }
begin
  r.AH:=05;
  r.CH:=bin2BCD(d.secol);
  r.CL:=bin2BCD(d.an);
  r.DH:=bin2BCD(d.luna);
  r.dl:=bin2BCD(d.data);
  intr($1a,r)
end;

procedure SetAlarm;             { seteaza alarma }
begin
  r.AH:=06;
  r.CH:=bin2BCD(t.ora);
  r.CL:=bin2BCD(t.min);
  r.DH:=bin2BCD(t.sec);
  intr($1a,r)
end;

procedure ResetAlarm;           { anulare alarma }
begin
  r.AH:=07;
  intr($1a,r)
end;
End.
```

Problema compatibilității cu anul 2000

Problema compatibilității Bios-ului cu anul 2000 înseamnă schimbarea corectă a datei odată cu schimbarea datei sistemului la schimbarea anului și mileniului din 31 decembrie 1999 la 1 ianuarie 2000 și recunoașterea și raportarea corectă a datei după anul 2000 (până în anul 2099!). Compatibilitatea la nivel Bios cu această problemă înseamnă recunoașterea corectă a datei numai de către sistem și nu garantează funcționarea sigură a programelor care folosesc date calendaristice (programe de gestiune, aplicații bancare, sisteme de control). Aplicațiile care folosesc date calendaristice trebuie testate și certificate separat pentru compatibilitatea cu această problemă (*year* 2000 *bug*) ceea ce duce la un volum imens de aplicații care trebuie modificate/certificate într-un timp relativ scurt.

Testarea Bios-ului pentru compatibilitate cu anul 2000 se poate face foarte simplu prin intermediul funcțiilor oferite de întreruperea `1ah`. Pentru aceasta se salvează data și ora curentă și se schimbă data la 31 decembrie 1999, ora 23:59:55. Urmărind ceasul puteți afla dacă trecerea la anul nou 2000 este făcută cu actualizarea corectă a datei. Programul care testează schimbarea datei în BIOS este prezentat mai jos.

Pentru **AwardBios** după data apariției se poate determina dacă este compatibil cu trecerea la anul 2000 astfel:

a) Dacă BIOS-ul are data producerii înainte de 26 aprilie 1994, trebuie să resetați ceasul sistem o singură dată astfel:
   1. opriți calculatorul înainte de miezul nopții pe 31 decembrie 1999;
   2. porniți-l cândva după miezul nopții, pe 1 ianuarie 2000;
   3. scrieți apoi data corectă în `Setup`.

b) Dacă BIOS-ul a fost produs între 26 aprilie 1994 și 31 mai 1995 trebuie să obțineți un Bios actualizat de la firma producătoare sau să resetați ceasul sistem zilnic!

c) Dacă BIOS-ul a fost produs după 31 mai 1995, calendarul face automat trecerea de la anul 1999 la anul 2000 la miezul nopții. Nu mai trebuie să faceți nimic, indiferent dacă sistemul este pornit sau oprit la miezul nopții.

Pentru detalii suplimentare pot fi utile și referințele următoare:

www.ami.com www.award.com
www.phoenix.com www.unicore.com

Listing: Y2K.PAS

```
program anul2000;
uses crt, rtc;
const anul_nou:data
          =(secol:19;an:99;luna:12;data:31);
      ora_12:timp=(ora:23;min:59;sec:55);
var azi, D:data;
    T:timp;
    i:Integer;

procedure HideCursor; assembler;
          { ascunde cursorul de pe ecran }
asm
  mov AX, $0100
  mov CX, $2607
  int $10
end;

procedure ShowCursor; assembler;
                { arata cursorul pe ecran }
asm
  mov AX, $0100
  mov CX, $0506
  int $10
end;

procedure ScrieData;          { afiseaza data }
var d:data;
begin
  ReadDate(d);
  with d do
  begin
    if data<10 then Write('0');
    Write(data,'/');
    if luna<10 then Write('0');
    Write(luna,'/',secol);
    if an<10 then Write('0');
    Write(an)
  end
end;
```

```
procedure ScrieOra;              { afiseaza ora }
var t:timp;
begin
  ReadTime(t);
  with t do
  begin
    if ora<10 then Write('0');
    Write(ora,':');
    if min<10 then Write('0');
    Write(min,':');
    if sec<10 then Write('0');
    Write(sec)
  end
end;

Begin
  HideCursor;
  TextBackGround(black);
  Clrscr;
  TextColor(yellow);
  GotoXY(2,25);
  Write('-= Program 2000 v. 1.0 - (c)');
  Write(' 1999 Tomsa Ion-Gelu =-');
  TextColor(cyan);
  GotoXY(1,3);
  Write(' Acest program testeaza daca ');
  Writeln('Bios-ul calculatorului dvs.');
  Write('actualizeaza corect data la ');
  Writeln('trecerea in anul 2000.');
  TextColor(magenta);
  Write('Apasati orice tasta pentru a ');
  Writeln('incepe testarea...');
  TextColor(cyan);
  ReadKey;
  Write('Memorez data si ora curenta: ');
  ScrieData;
  Writeln(' ora ');
  ScrieOra;
  ReadDate(azi);
  ReadTime(T);
  Write('Schimb data si ora: ');
  SetDate(anul_nou);
  SetTime(ora_12);
  i:=0;
  TextColor(white);
  GotoXY(wherex,wherey-1);
  while i<10 do
  begin
    GotoXY(21,WhereY+1);
    ScrieData;
    Write(' ora ');
    ScrieOra;
    Delay(1000);
    Inc(i)
  end;
  Writeln;
  ReadDate(d);
  if d.secol=20
  then
 Writeln('Actualizare corecta a datei BIOS !')
  else
  begin
    Write('Bios-ul calculatorului dvs. nu ');
   Writeln('este compatibil cu anul 2000...');
  end;
  TextColor(cyan);
  Write('Refac data si ora: ');
  SetDate(azi);
  Inc(t.sec,10);
  if t.sec>59
  then
  begin
    Inc(t.min);
    Dec(t.sec,60)
  end;
  SetTime(t);
  ScrieData;
  Writeln(' ora ');
  ScrieOra;
  TextColor(magenta);
  Writeln('Apasati orice tasta');
  Readkey;
  ClrScr;
  TextColor(cyan);
  Write('Program: ');
  TextColor(yellow);
 Writeln('2000 v.1.0(c)1999 Tomsa Ion-Gelu');
  TextColor(cyan);
  Write('Autor :');
  TextColor(LightBlue);
  Writeln('Tomsa Ion Gelu');
  TextColor(Cyan);
  Write('e-mail:');
  TextColor(Red);
  Writeln('tom@cnamd.ubbcluj.ro');
  Delay(5000);
  ShowCursor
End.
```

Bibliografie:

1. Vlad Căprariu, Andrei Enyedi, Marius Muntean, Marius Chiorean - *Sistemul de operare DOS. Ghidul programatorului*, Editura MicroInformatica, Cluj-Napoca 1991;
2. *Tech Help!*, manual electronic hipertext.

Domnul Tomșa Ion Gelu este profesor de informatică la Colegiul Național "Andrei Mureșanu", Dej. Poate fi contactat prin e-mail la adresa: tom@cnamd.ubbcluj.ro

CRIPTOGRAFIA în sisteme electronice de plăți

Victor-Valeriu Patriciu

Așa cum am promis în numerele anterioare ale gazetei, ne vom opri asupra unor aplicații ale sistemelor criptografice, simetrice sau cu chei publice, în serviciile de rețea. Vom începe cu unele dintre cele mai importante aplicații în contextul folosirii tehnologiilor informatice în comerț și e-comerț: sistemele de plăți electronice.

Extraordinara dezvoltare a interconectivității calculatoarelor în Internet, în toate segmentele societății, a condus la o tendință tot mai evidentă a companiilor de a folosi aceste rețele în aria unui nou tip de comerț, *comerțul electronic în Internet*, care să apeleze, pe lângă vechile servicii amintite și altele noi. Este vorba, de exemplu, de posibilitatea de a se efectua cumpărături prin rețea, consultând cataloage electronice "on" pe Web sau cataloage "off" pe CD-ROM și plătind prin intermediul cărților de credit sau a unor portmonee electronice. Pentru alții, comerțul Internet reprezintă relațiile de afaceri care se derulează prin rețea între furnizori și clienți, ca o alternativă la variantele de comunicații "tradiționale" prin fax, linii de comunicații dedicate sau EDI pe rețele cu valoare adăugată.

Acest nou tip de comerț a stimulat însă cererea pentru noi metode adecvate de plată. În cadrul noului concept, denumit sugestiv "*satul global*" (*global village*), dezvoltarea unor activități comerciale între participanți situați la mari distanțe geografice unii de alții nu poate fi concepută fără folosirea unor *sisteme electronice de plăți*. Aceste noi mijloace de plată permit transferarea comodă, sigură și foarte rapidă a banilor între partenerii de afaceri. De asemenea, înlocuirea monedelor și bancnotelor, actualele forme tradiționale de numerar, prin ceea ce denumim *bani electronici*, conduce, pe lângă reducerea costurilor de emitere și menținere în circulație a numerarului, și la o sporire a flexibilității și securității sistemelor de plăți.

Sistemele electronice de plăți trebuie să atingă niveluri ridicate de securitate, viteză, caracter privat și confidențial, descentralizare și internaționalizare și să fie unanim acceptate atât de cumpărători cât și de comercianți sau afaceriști. O trăsătură comună a majorității acestor soluții o constituie *utilizarea tehnicilor criptografice*, care asigură confidențialitatea, autenticitatea și integritatea mesajelor transferate între entitățiie implicate. O prezentare a metodelor criptografice clasice (cu cheie simetrică secretă) și a celor cu chei publice, am făcut în numerele trecute ale lui G-Info.

Cardul în sistemele electronice de plăți

Bank Americard (în prezent ***VISA International***), emis de Bank of America, reprezintă primul card emis în 1960 pentru revoluționarea modului de achiziționare de bunuri și servicii. În zece ani s-a ajuns la peste 20 milioane de deținători de astfel de carduri, în 1980 numărul a crescut la 73 milioane, iar în 1991 peste 100 milioane.

În anul 1966, 17 bănci finanțează ***Asociația Interbancară de Carduri,*** care se ocupa cu procedurile de autorizare, clearing și decontare. În 1979 ea devine ***Mastercard***, care cunoaște, la fel cu VISA, o dezvoltare rapidă: peste 90 de milioane de utilizatori în 1990.

De ce a cunoscut acest nou instrument de plată o așa de mare dezvoltare?

- Pentru ***deținătorii de carduri:*** ușurința în utilizare, renunțarea la purtarea numerarului, renunțarea la inconvenientele folosirii cecurilor, obținerea de credite instantanee, amânarea plăților cu circa o lună, sumele cheltuite se pot urmări cu ușurință la sfârșit de lună etc.
- Pentru ***comercianți:*** ușurința încasărilor, autorizarea simplă a vânzărilor, creșterea volumului vânzărilor, acordarea de fapt de vânzări pe credit la scară foarte mare.
- Pentru ***bănci:*** cardul reprezintă o modalitate simplă de a acorda credite și a încuraja clienții să se împrumute, veniturile băncii sporesc prin dobânzile încasate și prin comisioanele pecepute comercianților pentru serviciile de prelucare și decontare a tranzacțiilor cu carduri și pot avea clienți ce locuiesc la mari distanțe de bancă etc.

La nivel mondial, se folosesc următoarele ***sisteme de carduri mai cunoscute:***

- ***VISA International*** - este o organizație non-profit, alcătuită din organisme financiare cum ar fi bănci comerciale, case de economii etc. VISA este o marcă înregistrată, cu faimoasele benzi colorate în albastru, alb și galben și care permite decontarea tranzacțiilor internaționale. VISA domină actualmente piața internațională a cardurilor. Principalele tipuri de carduri VISA existente la ora actuală sunt ***VISA Classic, VISA Business, Gold/Premier, VISA Cash, VISA Travellers Cheques.***
- ***Europay International*** - este o organizație non-profit care cuprinde MasterCard International (cu sediul în SUA) și Eurocard International (cu sediul în Belgia). Se emit două tipuri de carduri: MasterCard și Eurocard. Principalele tipuri de carduri Europay existente la ora actuală sunt ***MasterCard/Eurocard Classic, Eurocard, Eurocheque, Maestro, Europay CLIP.***
- ***American Express*** - este un sistem global de plăți prin carduri, promovat de banca americană American Express. Principalele tipuri de carduri Europay existente la ora actuală sunt ***Business, Gold, Classic.***

Tipuri de carduri

Cardurile reprezintă instrumente de decontare care permit posesorului achiziționarea de bunuri și servicii, fără prezența efectivă a numerarului. Indiferent de tip și funcții, cardurile au anumite trăsături comune:

- Suport fizic de plastic, cu dimensiuni standard;
- Numele emitentului, numărul cardului, perioada de valabilitate, numele posesorului - inscripționate pe avers prin tipărire, gofrare sau gravare cu laser;
- Sigla emitentului, holograma de securitate, logo-ul organizației, pe avers;
- Banda magnetică pentru criptarea elementelor de securitate și un spațiu pentru semnătura posesorului, pe revers.

Cardurile se pot clasifica pe mai multe criterii:

A. În raport cu ***modul de stocare a caracteristicilor de securitate:***

- ***Cardurile cu bandă magnetică*** conțin pe 3 piste, informații criptate despre utilizator, emitent, algoritmul de codare etc;
- ***Cardurile cu microprocesor (inteligente) — smart-carduri —*** au stocate în chip datele de securitate și permit funcționarea ca portofel electronic.

B. În raport cu ***sursa de acoperire a cheltuielilor:***

- ***Debit-Cardurile*** asigură utilizatorului achiziționarea de bunuri sau servicii sau retrageri de numerar, cu condiția prezervării unor fonduri într-un *cont de card*, și efectuarea de cheltuieli în limita soldului disponibil; deși aceste carduri funcționează pe principiul lichidității existente în cont, băncile stimulează în mod "prudent" unele cheltuieli pe credit.
- ***Credit-Cardurile*** care asigură utilizatorului achiziționarea de bunuri sau servicii sau retrageri de numerar, pe baza unei linii de credit acordate posesorului de card. La rândul lor, ele pot fi de 2 tipuri:
  - ⇒ **Charge** - caz în care posesorul trebuie ca, în termen de circa o lună, să acopere depășirea de sold înregistrată sau creditul acordat;
  - ⇒ **Full** - cu funcțiuni depline, caz în care posesorul poate cheltui fără opreliști, în limita unui plafon sau credit negociat anterior.

C. În raport cu ***calitatea emitentului:***

- ***Carduri emise de bănci;***
- ***Carduri emise de societăți non-bancare*** (lanțuri de magazine, cluburi, parcaje etc).

În *domeniul plăților prin card*, transferul electronic de fonduri poate fi realizat cu ajutorul a *două tipuri de echipamente:*

- Puncte electronice de vânzare la comercianți (EFT POS-Point of Sale) reprezintă medii electronice de efectuare a tranzacțiilor comerciale, sub forma unor automate ce prelucrează în timp real autorizarea operațiunilor și stocarea informațiilor financiare și chiar decontarea tranzacțiilor. Autorizarea în timp real se asigură printr-o tastatură specială atașată la POS (numită PIN PAD) unde deținătorul cardului introduce numărul personal de identificare (PIN). La sfârșitul unei zile, memoria POS (în realitate un PC) se descarcă la bancă și astfel fondurile tranzacționate sunt înregistrate în contul comerciantului. Astăzi unele sisteme POS permit alimentarea instantanee a contului comerciantului, pe măsura efectuării vânzărilor. Operarea POS implică 2 carduri: al clientului și al vânzătorului (pentru identificarea acestuia).
- ***Ghișeul automat de bancă (ATM, cash dispenser...)*** este un dispozitiv care înlocuiește operatorul uman în operații, implicând cardul ca instrument de plată sau metodă de autentificare a clienților. Operarea ATM implică un card pe care utilizatorul îl introduce pentru identificare. Apoi acesta poate beneficia de anumite servicii: eliberarea de numerar din contul clientului, obținerea extrasului de cont, eliberarea de numerar din portofel electronic, alimentarea contului etc.

Sistem de plăți de tip POS

Vom analiza spre exemplu, modul în care poate funcționa un sistem care să permită utilizatorilor să plătească pentru bunuri și servicii prin *transfer electronic direct de fonduri* cu echipamente de tip *POS - Point of Sale.* Analiza se va face pe un sistem fictiv întrucât cele reale, din motive de securitate lesne de înțeles, nu sunt supuse publicării.

Sistemul constă din terminale, localizate la ieșirile din magazine care comunică cu *calculatorul băncilor* asociate. Fiecare utilizator (client) al sistemului are un **card** și un *număr personal de identificare (PIN),* eliberate de o bancă. Pe cardul din plastic este fie un strat magnetic fie o memorie integrată care conține datele personale ale clientului,

numărul de cont al posesorului și altele, printre care și data de expirare a cartelei. PIN-ul are tipic 4-6 cifre zecimale și reprezintă "semnătura" deținătorului cardului pentru autentificare; el nu trebuie să-l divulge unei alte persoane. În același timp deținătorul cardului trebuie să fie convins că PIN-ul este protejat în mod adecvat de autoritatea care-l știe.

Vom face ipoteza că la calculator există o bază de date completă pentru fiecare card emis de băncile asociate: informații asupra PIN-urilor și datelor de pe card, pentru toate cardurile emise. În acest fel se poate aproba orice tranzacție făcută cu card.

Când un client dorește să plătească bunuri sau servicii, prezintă cardul, care este citit pe terminal. Acesta furnizează date despre deținător și identifică tranzacția. Utilizatorul, separat, introduce PIN-ul. Vânzătorul introduce detalii ale cumpărăturilor apoi *terminalul comunică cu calculatorul pe baza unui protocol*, în urma căruia plata este aprobată sau nu.

La calculator se verifică dacă cartela este validă, dacă contul are bani pentru tranzacție și dacă PIN-ul și cartela corespund. Apoi terminalul informează calculatorul că tranzacția a fost operată. Este esențial ca:

- PIN-ul să fie ținut secret;
- mesajul să fie protejat împotriva unor eventuale schimbări în timpul transmisiei;
- cele două părți să se poată autentifica reciproc.

Aceste cerințe pot fi satisfăcute utilizând funcții criptografice bazate pe un cifru bloc ca DES, IDEA etc. Problema esențială nu este aceea a performanțelor algoritmului, cât a gestiunii cheilor de cifrare, care devine complicată, deoarece în sistem există sute de mii de terminale comunicând cu sute de calculatoare. Astfel, problema distribuirii și schimbării periodice a unei multitudini de chei criptografice este foarte grea. În consecință, dacă se utilizează cifrarea, este de dorit ca sistemul să conțină o procedură care să actualizeze automat cheile. Prin urmare este necesar ca sistemul să genereze chei noi pentru fiecare tranzacție de plată - numită *cheie de tranzacție*.

Pentru simplificarea descrierii, protocolul este prezentat prin fluxul de *mesaje cerere-răspuns* dintre terminalul POS și calculator. Protocolul începe după ce clientul a făcut cumpărăturile, a prezentat cardul în cititor la casă și a tastat PIN-ul, iar vânzătorul (casierul) a calculat valoarea bunurilor.

• Cum se generează cheia pentru o tranzacție de plată

Se folosește un principiu cunoscut, acela al punerii pe card și a unei date personale, dependente de proprietar - care să constituie așa-zisa *cheie card*. Această informație este citită, dar nu este transmisă. În plus, terminalul conține un *registru cheie* pentru fiecare bancă cu care este autorizat să comunice.

Să presupunem că o persoană este gata să cumpere și prezintă cardul de plastic la terminal. Terminalul citește *cheia card* și *numărul de cont al proprietarului* (NC) și identifică calculatorul (banca) pentru acel card. *Cheia de tranzacție (CT)* se generează apoi la terminal cu o funcție "one-way" aplicată cheii card și registrului acelui calculator. Dacă funcția "one-way" se bazează pe algoritmul DES, cheia de tranzacție se poate genera astfel:

$$\text{Cheia de tranzacție} = DES_{\text{registru cheie}}(\text{Cheie card}) \oplus \text{Cheie card}$$

Terminalul va trebui să-i furnizeze calculatorului suficientă informație pentru ca acesta să poată genera și el cheia de tranzacție. De aceea, *mesajul cerere* de la terminal va include în textul clar valoarea NC și identitatea terminalului.

Calculatorul menține un registru cheie pentru fiecare terminal POS din sistem (având conținut identic cu registrul cheie din terminal). De asemenea, calculatorul are o listă cu cheile card asociate fiecărui NC. În acest fel calculatorul este capabil să construiască cheia de tranzacție, identificând cheia card din NC și registrul cheie din identificarea terminalului. Prin urmare, cheia de tranzacție are un caracter aleator, determinat de *cheia card*. Actualizarea registrului cheie de la sfârșitul tranzacției va spori securitatea sistemului și va face ca fiecare *cheie de tranzacție* să fie diferită.

• Generarea mesajului cerere de către terminalul POS

După ce terminalul citește cardul și calculează cheia de tranzacție, se construiește un *mesaj cerere* (figura 1). El trebuie să conțină în clar:

- NC-ul proprietarului.
- Identificarea terminalului.

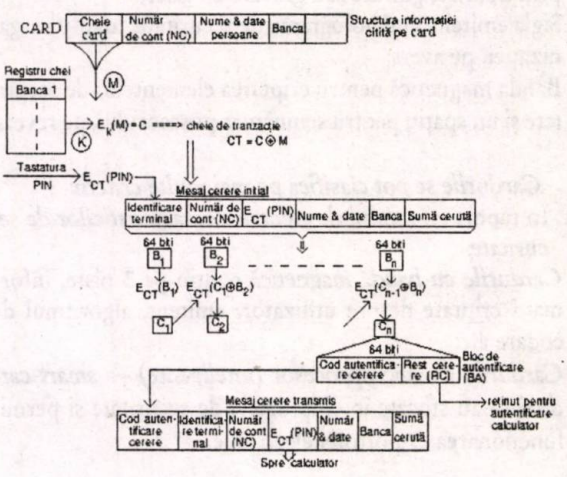


Fig. 1 Mesajul cerere

Se mai adaugă PIN-ul tastat de utilizator, cifrat cu *cheia de tranzacție*, precum și alte informații cifrate. După construirea mesajului cerere, terminalul trebuie:

- să-i adauge alte date care să-l protejeze împotriva unor schimbări;
- să conțină o procedură de autentificare a calculatorului;

• să conțină o cale de a verifica dacă răspunsul calculatorului se referă la mesajul cerere anterior emis.

Cele trei cerințe se ating prin generarea unui *bloc de autentificare (BA)* sub controlul cheii de tranzacție, în felul următor. Mesajul este divizat în n blocuri $B_1,B_2,...,B_n$, fiecare de 64 de biți. Cu ajutorul algoritmului DES se prelucrează pe rând fiecare bloc sub controlul *cheii de tranzacție*:

$$C_1 = DES_{CT}(B_1), C_2 = C_1 \oplus DES_{CT}(B_2),...,C_n = \\ = C_{n-1} \oplus DES_{CT}(B_n) = BA$$

Jumătatea din stânga a lui *BA* formează *codul de autentificare al cererii CACer*. Acesta este inserat în mesajul cerere și permite calculatorului să verifice că mesajul nu a fost schimbat în timpul transmisiei. Ceilalți 32 de biți ai lui *BA* formează *restul cererii (RC)*. Acesta este reținut în terminal și folosit ulterior pentru autentificarea *mesajului răspuns*.

• Verificarea mesajului cerere la calculator

Când calculatorul primește mesajul cerere, el va genera cheia de tranzacție, extrage *CACer* al cererii din mesaj; apoi generează cu același algoritm *BA*-ul pentru restul de mesaj (figura 2). Se verifică rezultatul obținut cu *CACer*-ul cererii pentru a se confirma integritatea mesajului. Dacă cele două valori coincid, se autentifică terminalul de către calculator. Partea dreaptă a *BA*-ului calculat este reținută pentru mesajul răspuns.

• Generarea mesajului răspuns de către calculator

Atunci când calculatorul a terminat de verificat elementele tranzacției, el va pregăti un *mesaj de răspuns* (figura 2) și va genera *BA*-ul sub controlul cheii de tranzacție. El va adăuga la mesajul de răspuns *restul cererii*, iar apoi va construi *BA*-ul pentru întreg mesajul extins. Acest lucru va permite terminalului să autentifice calculatorul ca fiind originea răspunsului și să confirme că răspunsul corespunde cererii emise.

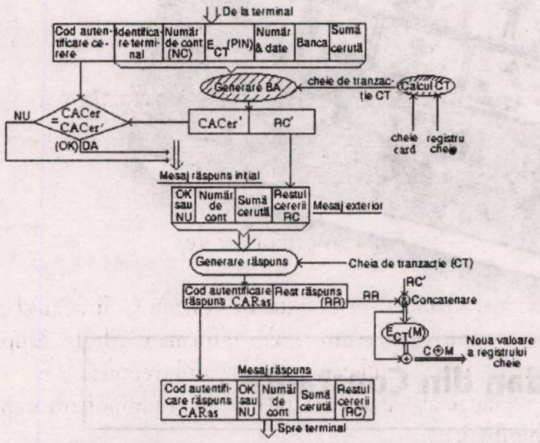


Fig. 2 Mesajul răspuns

Partea stângă a *BA*-ului de răspuns este codul de autentificare al răspunsului (*CARas*). Acesta este inserat în mesajul răspuns după ce restul cererii a fost șters și servește terminalului să autentifice mesajul. Cealaltă parte a *CARas*-ului este restul răspunsului (*RR*); el este reținut la calculator împreună cu restul cererii pentru actualizarea registrului cheie.

• Verificarea mesajului răspuns la terminalul POS

Când terminalul primește mesajul răspuns, scoate *CARas*-ul, adaugă restul cererii la începutul mesajului și generează *BA*-ul pentru mesajul extins. Partea stângă este comparată cu *CACer*-ul cererii obținut din mesajul de răspuns. Dacă acestea coincid, terminalul a identificat atât mesajul (integritatea lui) cât și calculatorul cu care comunică și confirmă că răspunsul primit corespunde la cererea de mesaj emisă. Apoi terminalul reține partea dreaptă a *BA*-ului ca restul răspunsului și încheie tranzacția (fig. 3)

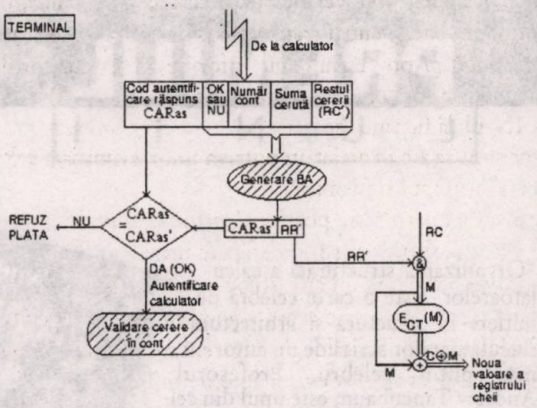


Fig. 3 Analiza răspunsului la terminaul POS

• Actualizarea registrului cheie pentru următoarele tranzacții

După ce calculatorul a transmis răspunsul, cheia de tranzacție este distrusă, se memorează valoarea curentă a registrului cheie și registrul cheie este actualizat la o nouă valoare, după următoarea procedură:

$$\text{Registru cheie} = DES_{\text{registru cheie}}(\text{rest cerere} \,||\, \text{rest răspuns}) \oplus \\ \text{rest cerere} \,||\, \text{rest răspuns}$$

În acest fel noua valoare a registrului cheie, depinzând de mesaje și de valoarea cardului, este nereproductibilă.

La terminal actualizarea registrului cheie se realizează identic. Dacă însă la terminal nu se confirmă mesajul sau apare o eroare pe linie, se ajunge la situația în care la calculator registrul e actualizat, pe când la terminal nu. De aceea se reține la calculator ultima valoare înscrisă în registrul cheie, pentru a se putea reface situația inițială.

În concluzie, se pot sublinia câteva caracterisitci ale sistemului:

- cheia de tranzacție este unică pentru o tranzacție de plată; ca urmare, chiar dacă cineva cunoaște cheia și cardul, nu poate deduce următoarea cheie, deoarece cheia depinde de card și de registrul cheie, care depinde la rândul său de cardul cumpărătorului precedent;

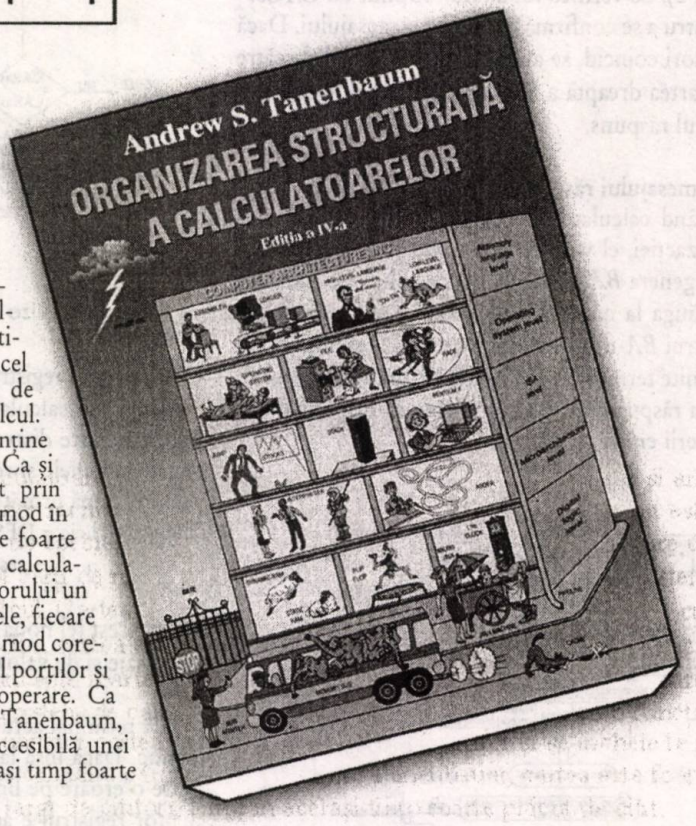
- chiar dacă cineva fură terminalul POS sau îl reproduce, nu îl poate folosi, deoarece pe lângă card trebuie cunoscut și PIN-ul; ca urmare nu e necesară o securitate fizică a terminalelor.

Inițializarea sistemului se poate face cu un card de test, pe care-l posedă vânzătorul și care precede toate tranzacțiile. Acest card stabilește prima valoare a registrului cheie și poate fi folosit și în caz de cădere a sistemului, la reinițializare.

Exemplul prezentat este doar una din nenumăratele aplicații pe care criptografia computațională le are în variate domenii conexe cu informatica. Sistemele electronice de plăți, instrumentul viitorului, care va înlocui incomodul și costisitorul numerar, este doar unul din acete domenii. În acest context, este suficient să amintim *cardurile inteligente cu microprocesor*, ce vor constitui adevărate portofele electronice, cu funcțiuni și cararcteristici de securitate mult superioare celor ce se folosesc astăzi.

Domnul prof. dr. Victor-Valeriu Patriciu este cadru didactic la Universitatea Tehnică Militară. Poate fi contactat prin e-mail la adresa: vip@mta.ro

PREMIUL LUNII



"Organizarea structurată a calculatoarelor" este o carte celebră de inițiere în structura și arhitectura calculatoarelor scrisă de un autor nu mai puțin celebru. Profesorul Andrew Tanenbaum este unul din cei mai faimoși autori de cărți de specialitate cunoscut în toate facultățile de calculatoare din lume. Cărțile sale sunt utilizate drept cărți de referință pentru cel puțin trei domenii: rețele, sisteme de operare și arhitectura sistemelor de calcul. Cartea a ajuns la ediția a patra și conține cele mai noi cunoștințe în domeniu. Ca și precedentele ediții cartea a rezultat prin rescrierea ediției precedente. În acest mod în noua ediție sunt prezente și concepte foarte noi cum sunt tehnologiile Java sau calculatoarele paralele. Conform viziunii autorului un sistem de calcul este o ierarhie de nivele, fiecare cu o funcționalitate precis definită. În mod corespunzător cartea pornește de la nivelul porților și se încheie la nivelul sistemelor de operare. Ca toate celelalte cărți ale profesorului Tanenbaum, cartea este foarte bine scrisă, fiind accesibilă unei categorii largi de cititori, fiind în același timp foarte plăcut de citit.

Câștigătorul este: Bâtă Bogdan din Constanța

Ada Byron... Niklaus Wirth...

De la FORTRAN la JAVA

Gabriela Stroe

Un limbaj de programare se definește ca fiind o notație sistematică prin care este descris un proces de calcul. Scopul creării unui astfel de limbaj este de a oferi programatorilor noi construcții sintactice pentru organizarea calculelor. Următoarele pagini vor descrie modul în care au evoluat aceste construcții sintactice de la primul limbaj de nivel înalt (FORTRAN) până la cea mai nouă tehnologie din domeniul IT&C (JAVA).

Prima generație

La mai puțin de zece ani după terminarea WW2 un grup de cercetători ai firmei IBM au căutat noi metode pentru efectuarea calculelor numerice. Acest grup, condus de *John Backus*, a studiat posibilitatea realizării unui așa numit *translator algebric*, un limbaj dedicat în principal acestor calcule numerice. Principala dificultate cu care s-a confruntat grupul condus de Backus nu a fost specificarea sau implementarea limbajului, ci ineficiența codului rezultat. Specificațiile pentru prima versiune (versiunea 0) a limbajului *FORTRAN* (*FORmula TRANslation*) au fost definitivate în anul 1954, primul compilator apărând în 1957. Tot în 1957 apar specificațiile pentru versiunile I și II ale limbajului. Popularitatea acestui limbaj este datorată, în mare măsură, sprijinului material oferit de firma IBM care a livrat gratuit produsul. Alte motive care au contribuit la impunerea limbajului sunt următoarele:

- este primul limbaj de programare de nivel înalt
- are o sintaxă simplă, apropiată de scrierea matematică, iar firma IBM oferă o bibliotecă de subprograme științifice extrem de puternică.

Versiunile *FORTRAN IV* (1966) și *FORTRAN77* (1977) au fost implementate pe o gamă variată de calculatoare începând cu microsistemele și terminând cu supercalculatoarele.

A doua generație

Unul din limbajele care au avut o contribuție remarcabilă la dezvoltarea limbajelor de programare este *ALGOL*. Prima versiune a apărut în 1958, dar versiunea de referință este *ALGOL60*, apariția sa reprezentând un moment important care a marcat întreaga dezvoltare a limbajelor de programare. Cea mai semnificativă dovadă în acest sens este "inventarea" termenului de *ALGOL-like* (de tip *ALGOL*), folosit atunci când se face o referință la genealogia unei familii de limbaje de programare și având înțelesul de limbaj care folosește majoritatea conceptelor introduse de *ALGOL60*. Impactul produs de acest limbaj s-a manifestat și în modul de descriere a algoritmilor prezentați în articole științifice și în cărți de specialitate. Cele mai importante caracteristici ale unui limbaj din familia *ALGOL-like*:

- sunt algoritmice (permit descrierea proceselor de calcul);
- sunt imperative (algoritmul este o secvență de modificări ale memoriei);
- au ca unități de bază *blocul* și *procedura*;
- conțin conceptele de *tip* și *verificare a tipului*;
- specificările sunt făcute prin reguli sintactice;
- necesită compilare.

Limbajul nu a cunoscut o utilizare pe măsura calităților sale. Câteva din motivele pentru care limbajul a avut succes numai în lumea științifică sunt:

- lipsa intrărilor și a ieșirilor standard;
- mecanismul de transmitere a parametrilor prin nume;
- lipsa sprijinului firmei IBM care nu avea interesul să promoveze un concurent serios pentru limbajul *FORTRAN*.

Insuccesul comercial - pe de o parte - și impactul favorabil asupra cercetării științifice teoretice - pe de altă parte - au dus la apariția unor noi versiuni ale limbajului, între care *ALGOL-W* și *ALGOL68*. Deși au existat numeroase încercări de îmbunătățire a limbajului, acesta dispare practic din punct de vedere al aplicațiilor scrise cu ajutorul său, rămânând doar un jalon important din punct de vedere teoretic.

Cel de-al treilea limbaj de nivel înalt - COBOL - a apărut la începutul anilor '60, apariția sa fiind legată de Departamentul Apărării Statelor Unite (*USDOD - United States Department of Defence*) care a ajuns la concluzia că

este necesar un limbaj unic care poate fi utilizat în toate domeniile militare de activitate. A fost creat un grup de specialiști însărcinați cu proiectarea acestuia, rezultatul activității acestui grup materializându-se în ***raportul COBOL*** (*COmmon Business Oriented Language*) care a fost publicat în 1959. Un an mai târziu existau deja două compilatoare ***COBOL***. Datorită sprijinului primit din partea guvernului Statelor Unite și a constructorilor americani de calculatoare, limbajul se impune repede pe piață și devine cel mai utilizat limbaj de programare al deceniului șapte. Noile concepte teoretice introduse de ***COBOL*** în domeniul limbajelor de programare au o importanță majoră. La fel ca și limbajul ***FORTRAN***, ***COBOL-ul*** este perfecționat continuu, în 1981 apărând versiunea ***COBOL81*** care încerca să oprească ofensiva ***SGBD-urilor*** (*Sisteme de Gestiune a Bazelor de Date*).

În anii '60 a avut loc o adevărată revoluție în domeniul limbajelor de programare, în acest câmp de activitate putându-se distinge câteva tendințe importante. Limbajele ***FORTRAN*** și ***COBOL*** devin limbaje standard pentru aplicații științifice și comerciale, apar mai multe limbaje ***ALGOL-like***, se încearcă înlocuirea vechilor limbaje cu altele noi care să păstreze caracteristicile pozitive ale predecesorilor și apar limbaje bazate pe concepte noi. Linia ***ALGOL60*** este continuată de ***ALGOL-W*** și ***EULER***, ambele fiind dezvoltate de *Niklaus Wirth*. ***ALGOL-W*** introduce o nouă metodă de transmitere a parametrilor numită ***metoda valoare-rezultat***, precum și instrucțiunea *case*. ***EULER*** încearcă tratarea procedurilor ca obiecte și folosește precedența simplă ca tehnică de analiză sintactică. Un alt limbaj bazat pe ***ALGOL*** este ***SIMULA67***, realizat în Norvegia, și care a fost conceput pentru aplicații în domeniul simulării. Acest limbaj este primul care introduce noțiunea de ***clasă***, noțiune cu semnificația de grup de declarații și metode care sunt tratate împreună, ca unitate distinctă de program. Obiectele clasei pot fi generate dinamic și au o durată de viață independentă de locul în care au fost create. Acest nou concept este considerat a fi părintele tipurilor abstracte de date.

Limbajul ***APL*** (*A Programming Language*) a fost realizat de *Kenneth Iverson* pentru firma IBM. Scopul autorului a fost realizarea unui limbaj puternic, care să utilizeze o notație compactă pentru descrierea operațiilor matematice. Prima utilizare remarcabilă a limbajului a fost folosită la descrierea familiei de calculatoare IBM/360. Deși prima implementare a fost făcută destul de târziu, astăzi există multe implementări pe diverse tipuri de calculatoare, cu toate că unele caracteristici ale sale contravin regulilor consacrate ale teoriei limbajelor de programare.

Limbajul ***LISP*** (*LISt Processing*) a fost realizat de *John McCarthy* la ***MIT*** (*Massachusetts Institute of Technology*) pentru a fi folosit în domeniul inteligenței artificiale. Acest limbaj introduce un nou tip de programare numit ***programare funcțională***. În ***LISP*** datele și expresiile sunt prezentate uniform, sub forma *S-expresiilor* (*symbolic-expressions*); apare o formă nouă a expresiei condiționale; se utilizează forma prefix a operatorilor; structura fundamentală de control este *recursivitatea* și este preferată tehnica ***culegerii resturilor*** (garbage collection) în locul ștergerii explicite a referințelor. La peste 30 de ani de la crearea sa, ***LISP*** rămâne în actualitate nu numai datorită calităților sale dar, mai ales, datorită dezvoltării inteligenței artificiale, principalul domeniu în care este folosit.

Limbajul ***SNOBOL*** a fost creat la Laboratoarele Bell de către *Farber, Polonski* și *Griswold*. Cea mai importantă calitate a acestui limbaj este prelucrarea șirurilor de caractere. La începutul anilor '80 a apărut versiunea ***SNOBOL4*** care conținea un mecanism de macrodefiniții foarte eficient.

A treia generație

Limbajul ***PL/1*** a apărut ca urmare a dorinței firmei IBM de a realiza limbaje noi care să le înlocuiască pe cele existente. Acest limbaj era "pregătit" pentru noua familie 360 care era lansată în aceeași perioadă, el preluând numeroase caracteristici ale predecesorilor săi, printre care instrucțiunea de atribuire din ***FORTRAN***, structura de bloc și procedurile recursive din ***ALGOL60***, structurile de date din ***COBOL***. ***PL/1*** introduce câteva concepte noi cum ar fi ***gestiunea excepțiilor*** care permite programatorului să prevadă secvențe speciale de program pentru tratarea situațiilor deosebite care pot apărea în timpul execuției și ***multitasking-ul*** care are un rol major în descrierea paralelismului.

Spre sfârșitul deceniului șapte apare raportul ***ALGOL68*** care introduce două concepte noi: ***generalitatea*** și ***ortogonalitatea***. Între membrii grupului care a realizat acest raport s-a aflat și *Niklaus Wirth* care, nefiind de acord cu publicarea raportului, s-a retras și apoi a realizat limbajul ***PASCAL***. Ideea de bază urmărită în proiectarea acestui limbaj a fost operarea cu un număr mic de concepte integrate care să fie compatibile într-un cod eficient. Cele mai importante calități ale limbajului sunt:

- mecanismul de structurare a datelor care permite un nivel superior de abstractizare;
- existența unei definiții axiomatice a limbajului;
- mecanismul de verificare a programelor în faza de compilare.

Primul compilator ***PASCAL*** a fost realizat tot de Wirth în 1971. De atunci acest limbaj este preferatul mediilor academice, având o largă utilizare datorită ușurinței cu care se pot realiza programe puternice și eficiente utilizându-se un set redus de concepte. Limbajul ***PASCAL*** este folosit deseori ca limbaj de descriere a algoritmilor înlocuind predecesorul său, ***ALGOL***.

Generația a patra

În deceniul opt, accentul s-a pus pe îmbunătățirea stilului de programare în limbajele imperative. Se observă următoarele trei tendințe majore:

- dezvoltarea conceptului de ***tip abstract de date***, apărut o dată cu conceptul de clasă în ***SIMULA*** și dezvoltat în această perioadă în cadrul limbajelor ***CLU***, ***EUCLID***, ***MODULA*** și ***ADA***. Acest concept este tratat atât din

punct de vedere al proiectării (enunțării specificațiilor) unui limbaj de programare, cât și ca o proprietate a unui limbaj de programare, în sensul de facilitate oferită programatorului;

- perfecționarea mecanismelor de tratare a excepțiilor, limbajele *ADA* și *CLU* fiind în acest sens, exemple semnificative;
- încorporarea de mecanisme pentru descrierea proceselor paralele. Pentru sincronizarea proceselor s-au dezvoltat mai multe concepte cum ar fi ***semaforul***, ***monitorul*** (supraveghetorul) și ***sincronizarea prin transmiterea de mesaje***. Limbajele care oferă facilități privind concurența sunt *MODULA* și *ADA*. Limbajul C este unul dintre cele mai importante limbaje apărute în acest deceniu, el fiind folosit pentru scrierea sistemului de operare *UNIX*.

La aproape două decenii după apariția limbajului *COBOL*, istoria se repetă. Departamentul Apărării Statelor Unite constată că pe calculatoarele folosite în domeniul militar sunt rulate programe scrise într-un număr mare de limbaje diferite și consideră oportună crearea unui nou limbaj de programare destinat sistemelor hibride de calcul (*embedded computer systems*). Aceasta a fost modalitatea prin care s-a ajuns la limbajul *ADA*. Primele patru proiecte ale specificațiilor au purtat denumirile de *STRAWMAN* (*omul de paie*), *WOODENMEN* (*omul de lemn*), *TINMAN* (*omul de tinichea*) și *IRONMAN* (*omul de fier*) și au fost elaborate de specialiști ai Departamentului Apărării. Ultima versiune - *IRONMAN* - a fost supusă competiției între diverse companii specializate în elaborarea specificațiilor, câștigătoare fiind firma CII-Bull din Franța. Noul set de specificații, definitivat de un grup condus de *Jean Ichbiah*, a primit numele de *STEELMAN* (*omul de oțel*), iar limbajul rezultat a fost denumit *ADA*, în cinstea ***Adei Augusta***, contesă de Lovelace, fiica poetului Byron, cea care este considerată prima programatoare din istorie. Publicarea noilor specificații a avut loc la începutul anilor '80. *ADA* este un limbaj asemănător cu *PASCAL* care conține câteva facilități suplimentare pentru calculele numerice, intrările și ieșirile nestandard, specificarea dependenței de mașină, gestiunea excepțiilor, abstractizarea datelor și concurență. În concordanță cu specificațiile, proprietățile noului limbaj acoperă toată gama conceptelor moderne: ***modularitate***, ***portabilitate***, ***extensibilitate***, ***abstractizare***, facilități pentru dezvoltarea de programe și facilități pentru întreținere. După apariția specificațiilor, Departamentul Apărării a constituit un comitet care avea sarcina de a valida compilatoarele *ADA* realizate de diverși producători de software.

Generația actuală

Ultimii zece - cincisprezece ani au însemnat o etapă calitativ superioară din punct de vedere al dezvoltării informaticii. Direcțiile esențiale în ceea ce privește acest domeniu au fost:

- *dezvoltarea microelectronicii* care duce la apariția unor calculatoare mai compacte și mai rapide;
- *dezvoltarea arhitecturii calculatoarelor*;
- *apariția supercalculatoarelor*;
- *extinderea gamei de aplicații abordabile*;
- *apariția calculatoarelor personale* care duce la "democratizarea" informaticii, permițând accesul utilizatorului nespecialist la resursele unui sistem de calcul.

Efectele acestei situații pot fi observate în industria constructoare de calculatoare, industria software și gestiunea informației. În domeniul limbajelor de programare se observă o multitudine de realizări în diferite domenii:

- *inteligența artificială* (apare limbajul *PROLOG*);
- *programarea orientată pe obiect;* apare limbajul *SMALLTALK*, precum și versiuni orientate pe obiect ale limbajelor *PASCAL* și *C*;
- apariția de *noi standarde* pentru limbajele vechi;
- dezvoltarea de *limbaje de manipulare a datelor* pentru aplicațiile de gestiune, cu o sintaxă simplă, destinate utilizatorilor neprofesioniști; apar limbajele *DBASE*, *FOXBASE*, *REFLEX*, *PARADOX*, *FOXPRO*;
- apariția unor *instrumente software puternice*, destinate atât programatorilor experimentați, cât și utilizatorilor obișnuiți;
- dezvoltarea unor *pachete de programe științifice* puternice care înglobează funcții de gestiune a bazelor de date, prelucrare statistică și reprezentare grafică a rezultatelor obținute.

La momentul actual marea majoritate a conceptelor evoluate, vehiculate în cadrul limbajelor de programare se regăsește, sub diverse forme, în cadrul limbajului *C++*. Chiar dacă există argumente împotriva acestei afirmații, *C++* este cel mai utilizat limbaj de programare în proiectarea aplicațiilor de anvergură.

Generația următoare

Deși pare cam ciudat, generația următoare de limbaje de programare a apărut deja. Mulți spun că *JAVA* este limbajul mileniului următor, dar mai mult, acest limbaj este folosit deja pe o scară din ce în ce mai largă. Limbajul *JAVA* este creat de un grup de cercetători ai firmei Sun Microsystems (în numărul 8/2 al GInfo este prezentată istoria "plină de ciudățenii" a acestui limbaj) și lansat în anul 1995 deși prima implementare a fost realizată în 1991. *JAVA* (care a mai purtat și numele de *OAK*) este considerat cel mai potrivit limbaj pentru programarea în Internet datorită portabilității sale. În principiu un program scris în acest limbaj poate fi rulat pe orice platformă pentru care există un interpretor *JAVA*.

Bibliografie

1. ***Bazil Pârv, Alexandru Vancea*** - *Fundamentele limbajelor de programare*, Editura Albastră, Cluj-Napoca, 1996
2. ***Gazeta de Informatică*** - martie 1998

Gabriela Stroe este elevă în clasa a XII la Liceul "Mihai Eminescu" din Cluj-Napoca. Poate fi contactată prin intermediul redacției.

EASTER Eggs

Gabriela Stroe

Majoritatea programatorilor introduc în produsele lor unele "surprize" care își fac apariția în momentul în care utilizatorul efectuează anumite operații. Aceste "surprize" sunt denumite Easter Eggs, adică, în traducere "liberă și democratică", Ouă de Paște. Scopul introducerii acestora este, de obicei, acela de a face cunoscuți autorii produsului, deoarece majoritatea marilor firme de software "uită" să amintească numele celor care au realizat respectivul produs. Din această cauză, majoritatea programatorilor utilizează această metodă neortodoxă de a se face cunoscuți.

Deși în majoritatea cazurilor Easter Eggs-urile nu conțin decât numele și eventual poza realizatorilor programului, alteori putem avea și alte surprize. Vă prezentăm în continuare câteva Easter Eggs-uri "descoperite" până în prezent. Dacă aveți cunoaștinți de alte "surprize" de acest gen v-am ruga să ni le împărtășiți și nouă din cunoștințele dumneavoastră din acest domeniu, trimițând pe adresa redacției o descriere. Astfel, s-ar putea ca într-unul din numerele următoare să realizăm o continuare a acestui articol. Vă mulțumim anticipat!

Și acum iată cum puteți să vedeți și dumneavoastră astfel de surprize:

Adobe Premiere

- Alegeți opțiunea *About* și ținând apăsată tasta CTRL efectuați un click pe *Adobe*.

Ami Pro

- Din meniul *Help* selectați comanda *About*.
- Țineți simultan apăsate tastele CTRL, ALT și SHIFT.
- Apăsați tasta F7.
- Apăsați consecutiv literele S, P, A și M.
- Apăsați tastele corespunzătoare ultimei cifre și antepenultimei cifre din numărul indicat în *Available Memory*; de exemplu dacă numărul ar fi 2457 ar trebui să tastați 7 și 4.
- Vor apărea mici imagini cu echipa Ami Pro. Dacă veți efectua un click pe una din imagini aceasta va dispărea, excepție făcând una dintre ele (aleasă aleator). Dacă veți reuși să descoperiți această imagine veți avea o uriașă surpriză.

Ami Pro 3.0

- Deschideți meniul *Extras / Settings*.
- Apăsați tastele G, P și B.
- Selectați comanda *About*.
- Țineți simultan apăsate tastele CTRL, ALT și SHIFT.
- Apăsați tasta funcțională corespunzătoare zilei din săptămână potrivit datei sistemului. De exemplu pentru joi veți apăsa F4, pentru sâmbătă F6, etc.
- Tastați cuvântul SHANK.
- Eliberați tastele.

Beavis & Butthead Windows 95

- Deschideți meniul *Options* și tastați cuvântul SUCKS.

Borland C++ 4.0

- Din meniul *Help* selectați comanda *About*.
- Țineți simultan apăsate tastele ALT și I.

Borland DB Desktop V5.1

- Din meniul *Help* selectați comanda *About*.
- Apăsați tasta I.
- Apăsați simultan tastele SHIFT și Z.

Business Works v10

- Din meniul *Help* selectați comanda *About*.
- Tastați cuvântul MST3K.
- Efectuați un double-click pe iconița *BusinessWork*.
- Închideți fereastra de dialog care apare.
- Efectuați un nou double-click pe iconița *BusinessWork*.
- Numele membrilor echipei de realizare vor apărea defilând ca într-un generic de film.

Compton's Interactive Encyclopedia 1996

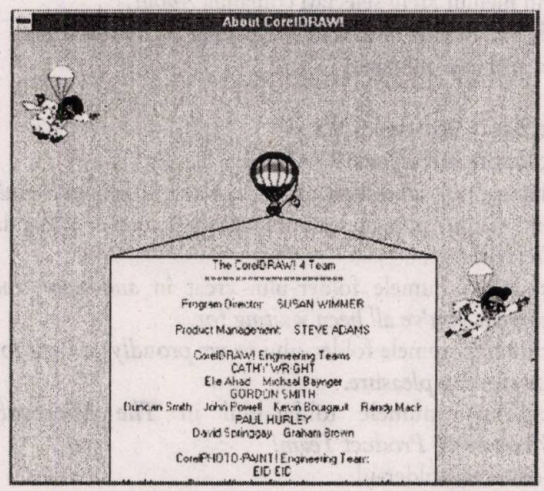
- Deplasați-vă deasupra oricărei ferestre de listare.
- Selectați al treilea articol.
- Efectuați un click.
- Introduceți de trei ori litera Z.

Corel Draw 3.0

- Ținând apăsate tastele **CTRL** și **SHIFT** selectați comanda *About* din meniul *Help*.
- Efectuați un **double-click** pe *balon*.
- În timp ce ții apăsat butonul drept al mouse-ului balonul va fi alimentat.

Corel Draw 4.0

- Din meniul *Help* selectați comanda *About*.
- Efectuați un **double-click** pe *balon* în colțul din stânga-sus.
- Fereastra va crește în dimensiune iar balonul Corel va arbora steagul canadian.
- Pentru a activa arzătorul balonului țineți apăsat butonul stâng al mouse-ului.
- În timp ce țineți butonul apăsat balonul se va înălța.
- Apăsați și butonul drept al mouse-ului.
- Din balon va fi lansat un Elvis cu parașută.

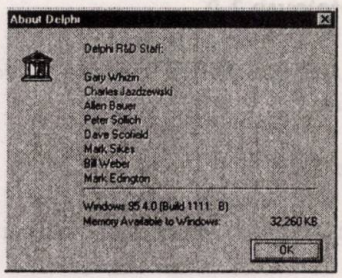


Corel Draw 5.0

- Lansați în execuție una din aplicațiile Corel.
- Din meniul *Help* selectați comanda *About*.
- Efectuați un **double-click** pe una dintre iconițe.
- Va apărea balonul Corel și o pârghie.
- Țineți apăsat butonul drept al mouse-ului.
- Balonul se va înălța cu un steag.
- Efectuați un **right-click**.
- În funcție de ziua și ora la care efectuați operațiile efectul va fi diferit, dar întotdeauna distractiv.

Delphi

- Din meniul *Help* selectați comanda *About*.
- Ținând apăsată tasta **ALT** tastați unul din cuvintele: **VERSION**, **TEAM**, **AND** sau **DEVELOPERS**.
- În funcție de cuvântul tastat veți putea vedea:
  - ♦ numărul versiunii interne;
  - ♦ o prezentare a echipei;
  - ♦ o animație;
  - ♦ echipa de realizare a programului.



Encarta 96 World Atlas

- Din meniul *Find* selectați comanda *Place*.
- La opțiunea *goto* introduceți **Necropolis of Mir**.
- Apăsati tasta **ENTER**.
- Folositi unealta *Zoom* pentru a vă putea duce la **509** km (introduceți manual valoarea).
- Efectuați un **click** pe o poză din fereastră.
- Veți vedea o poză a echipei de realizare.

Excel 4.0

- Deschideți un nou *Worksheet* cu bara de unelte standard (*Standard Tool Bar*) deasupra.
- Efectuați un **right-click** pe bara de unelte standard.
- Din meniul pop-up care va apărea selectați comanda *Customize*.
- Selectați *Custom*.
- Efectuați un **drag and drop** cu iconița *Solitaire* până într-un loc liber din *toolbar*.
- Efectuați un **click** pe butonul *OK* de sub *Assign to Tool* și apoi un **click** pe butonul *Close*.
- Ținând apăsate tastele **CTRL**, **ALT** și **SHIFT** efectuați un **click** pe iconița *Solitaire*.

Excel 7

- Deschideți un nou *Spreadsheet*.
- Coborâți până la linia *95*.
- Selectați întreaga linie prin efectuarea unui **click** pe numărul *95* din dreapta.
- Apăsați tasta **TAB** pentru a vă deplasa pe coloana *B*.
- Din meniul *Help* selectați comanda *About Microsoft Excel*.
- În timp ce țineți apăsată o tastă **CTRL** și ambele taste **SHIFT** efectuați un **click** pe *Tech Support*.
- Se va deschide o fereastră în care vă puteți deplasa. Pentru deplasare folosiți tastele arătătoare și tastele **C** și **D** pentru a vă deplasa în jos și în sus.
- Întoarceți-vă spre scări, departe de zid și tastați cuvântul **EXCELKFA**.
- Zidul va dispărea și veți putea intra într-o cameră cu poze.

Intuit Quicken 3.0

- Ținând apăsată tasta **S** din meniul *Help* selectați comanda *About*.
- Va apărea un surfer și numele programatorilor.

Logitech Fototouch 1.0
- Din meniul *Help* selectați comanda *About*.
- Ținând apăsată tasta SHIFT efectuați de două ori un click pe desen.

Lotus Notes 4.0
- Din meniul *Help* selectați comanda *About*.
- Tastați unul din textele **release the power** sau **elvis is not dead**.

Lotus Organize 1.0
- Ținând apăsate tastele CTRL, ALT și SHIFT efectuați un double-click pe coșul pentru hârtie.

MKS Source Integrity 7.1
- Din meniul *Help* selectați comanda *About*.
- Efectuați un double-click pe *logo*-ul programului.

Microsoft Acces 1.0 și Microsoft Acces 2.0
- Deschideți o bază de date.
- Creați o nouă tabelă numită **cirrus**.
- În tabela nou-creată creați un câmp fără cheie primară.
- Salvati tabela.
- În lista de tabele deplasați-vă pe **cirrus**.
- Din meniul *Help* selectați comanda *About Microsoft Acces*.
- Ținând apăsate tastele CTRL și SHIFT efectuați un right-double-click pe *logo*-ul programului.

Microsoft Acces 95
- Creați o nouă bază de date.
- Creați un *do-nothing macro* (macro care nu face nimic) și denumiți-l **Magic Eight Ball**.
- Efectuați un drag and drop cu macroul nou creat până pe *toolbar* astfel încât să fie creat un buton pentru el.
- Efectuați un right-click.
- Din meniul context care va apărea selectați comanda *Customize*.
- Efectuați un nou right-click.
- Din meniul context care va apărea selectați comanda *Choose Button Image*.
- Selectați *Magic Eight Ball*.
- Efectuați un click pe butonul *OK* și apoi un click pe butonul *Close*.

Microsoft Automap
- Ținând apăsată tasta SHIFT din meniul *Help* selectați comanda *About*.
- Va apărea o mașină roșie și alte "efecte speciale".

Microsoft Diagnostics
- Lansați în execuție programul MSD.EXE.
- Din meniul *Help* selectați comanda *About*.
- Apăsați tasta F1.

Microsoft Office 95
- Deschideți *toolbar*-ul din Microsoft Office 95.
- Efectuați un click pe *About Microsoft Office*.
- Ținând apăsate tastele CTRL și SHIFT efectuați un double-click pe *logo*-ul programului.
- Efectuați un click pe *ecranul cu titlu* care a apărut.

Microsoft PowerPoint 95
- Din meniul *Help* selectați comanda *About*.
- Efectuați un click pe iconița din stânga.

Microsoft Visual C++ 2.0
- Din meniul *Help* selectați comanda *About*.
- Ținând apăsată tasta CTRL efectuați un double-click pe grafic (tasta trebuie eliberată în același timp cu butonul mouse-ului).
- Folosiți **/a/z** pentru a zbura.

Microsoft Visual C++ 2.2
- Introduceți CD-ul *Visual C++* în unitatea CD.
- Lansați *Visual C++*.
- Din meniul *Help* selectați comanda *About*.
- Ținând apăsată tasta CTRL efectuați un double-click.
- Va apărea o animație.

Microsoft Windows 95 și Microsoft Windows 95 OSR2
- Efectuați un right-click într-o zonă liberă a *Desktop*-ului.
- Din meniul pop-up care va apărea selectați comanda *New/Folder*.
- Schimbați numele folder-ului creat în ***and now, the moment you've all been waiting for.***
- Schimbați numele folder-ului în ***we proudly present for yor viewing pleasure.***
- Schimbați numele folder-ului în ***The Microsoft Windows 95 Product Team!***
- Deschideți folder-ul.
- Veți vedea numele celor care au contribuit la realizarea sistemului de operare.

Netscape
- În căsuța *location* tastați *About:* urmat de unul din cuvintele *photo, authors, mozilla* sau **logo.**
- În funcție de cuvântul tastat va apărea:
  - ♦ o poză cu programatorii;
  - ♦ autorii;
  - ♦ puțină literatură;
  - ♦ sigla Netscape.
- Pentru a vedea Casa Albă tastați simultan CTRL, ALT și W.
- Pentru a vedea câteva statistici tastați simultan CTRL, ALT și W.

Norton Antivirus 1.5 pentru DOS
- Din meniul *Help* selectați comanda *About*.
- Apăsați simultan tastele SHIFT și F1.

Norton Antivirus pentru Windows 95
- Din meniul *Help* selectați comanda *About*.

- Apăsați simultan tastele N, A și V.
- Va apărea o fereastră cu pozele echipei care a realizat Norton Antivirus.

Norton Desktop 2.0

- Ținând apăsate tastele N, D și W din meniul *Help* selectați comanda *About*.
- Efectuați un **triple-click** pe iconița *Symantec*.

Norton Utilities 8.0

- Deschideți orice aplicație Norton Utilities.
- Din meniul *Help* selectați comanda *About*.
- Apăsați simultan tastele N, U și 8.
- Mișcați mouse-ul în fereastra cu poza realizatorilor.

Page Maker 4.0, 5.0 și 6.0

- Ținând apăsate tastele SHIFT și CTRL din meniul *Help* selectați comanda *About PageMaker* sau comanda *About TableEdit*.

PC Tools for Windows

- Din meniul *Help* selectați comanda *About*.
- Ținând apăsate tastele SHIFT și CTRL efectuați un **left-right-click** pe iconiță.

Quicken for Windows

- Pentru a afla versiunea pe care o dețineți ținând apăsată tasta SHIFT selectați comanda *About* din meniul *Help*.
- Pentru a vedea animația *șarpe cu clopoței* din meniul *Help* selectați comanda *About* și apăsați tasta R.
- Pentru a vedea animația *credits* din meniul *Help* selectați comanda *About* și apăsați tasta S.
- Pentru a vedea *informații despre versiunea extinsă* din meniul *Help* selectați comanda *About* și apăsați tasta V.
- Pentru a vedea *logo-ul animat* din meniul *Help* selectați comanda *About* și apăsați tasta R.

QuickTime for Windows 95

- Deschideți control panel-ul *QuickTime 32*.
- Efectuați un **click** pe *logo*-ul QuickTime.
- Efectuați un **right-click** pe *logo*-ul din fereastra *About* care va apărea.
- Va apărea o poză cu programatorii.

Simply Tax 95

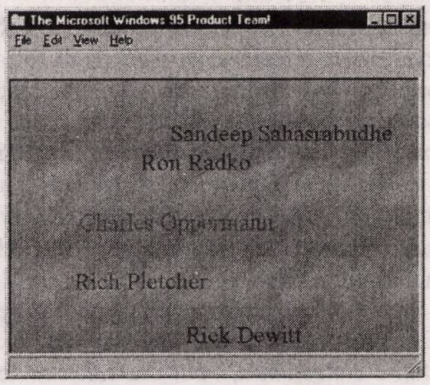
- Din meniul *Help* selectați comanda *About*.
- Efectuați un **double-click** pe iconiță.
- Tastați cuvântul **nimda**.

Turbo Pascal for Windows

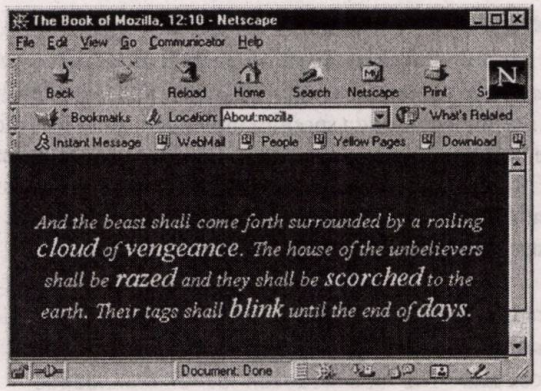
- Din meniul *Help* selectați comanda *About*.
- Apăsați simultan tastele **ALT** și **T**.

Cam atât cunoaștem noi despre Easter Eggs-urile din cele mai cunoscute programe. Majoritatea jocurilor conțin diferite astfel de surprize care permit gamer-ului să obțină diferite facilități. În "limbaj de specialitate" acest fel de surprize sunt numite **CHEAT**-uri. Mai există și unele jocuri serioase care includ și Easter Eggs-uri veritabile cum ar fi **Heroes of Might and Magic II** produs de firma **New World Computing**. Aici, dacă veți efectua un **click** pe ușa care se vede în ecranul de început al jocului și dacă nu ați șters fișierele cu animații, veți putea urmări secvențe din jocul **Might and Magic VI**.

Iată o captură a Easter-Egg-ului din Windows 95:



Puteți vedea mai jos și literatura din Netscape:



Mai trebuie precizat faptul că Easter Eggs-urile nu funcționează întotdeauna și pot fi schimbate la versiuni sau subversiuni mai noi.

Gabriela Stroe este elevă în clasa a XII-a la Liceul "Mihai Eminescu" din Cluj-Napoca, poate fi contactată prin intermediul redacției.

Pagini WEB cu HTML 4.0

Mihai Scorțaru

În acest episod vom încheia prezentarea generală a limbajului HTML. Veți afla mai multe informații despre modelele de stiluri introduse de specificațiile CSS și diferite artificii utile pentru imbunătățirea calității paginilor dumneavoastră. Serialul despre crearea paginilor Web nu se va încheia aici, deoarece există multe alte domenii care pot fi cercetate cum ar fi: limbajul VRML, noua specificație XML 1.0 care, probabil, se va impune ca standard pentru crearea paginilor Web, precum și limbajul SMIL pentru crearea de prezentări multimedia pe Web.

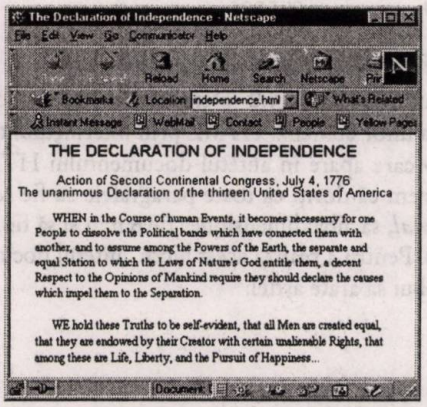
Implementarea suportului pentru specificațiile *CSS* (*Cascading Style Sheets*) a reprezentat o adevărată revoluție în crearea paginilor Web. Dintre toate noutățile și modificările introduse de specificațiile HTML 4.0 nici una nu este mai importantă și mai utilă decât ***modelele de stiluri***. În cadrul versiunilor HTML anterioare existau numai indicații de formatare cum ar fi tag-urile **<B>**, **<H2>** etc., iar browserul decidea (în funcție de felul în care a fost configurat) modul în care informațiile vor apărea pe ecran. Modelele de stiluri permit proiectanților de pagini Web să definească anumite caracteristici pentru orice tag HTML. Pot fi astfel definite *dimensiunea*, *tipul* sau *culoarea* fontului, precum și multe alte caracteristici.

Modele de stiluri

Cu ajutorul modelelor de stiluri poate fi modificată *spațierea liniilor*, o *margine* a textului fără modificarea celeilalte etc... O modalitate de a specifica modificări aduse stilurilor este folosirea atributului `STYLE` al diferitelor comenzi HTML cu condiția ca atributul `TYPE` al aceleiași comenzi să aibă valoarea `text/css`. În următorul exemplu va fi ilustrat modul de folosință al acestor două atribute:

```
<html>
<head>
  <title> The Declaration of Independence
  </title>
</head>
<body>
  <p type="text/css"
     style="font-size:    14pt;
            font-family: sans-serif;
            font-weight: bold;
            text-align:  center">
THE DECLARATION OF INDEPENDENCE
  <p type="text/css"
     style="font-size:    10pt;
            line-height: 10pt;
            font-family: helvetica;
            text-align:  center">
Action of Second Continental Congress, July 4,
1776 <BR>
The unanimous Declaration of the thirteen
United States of America
  <p type="text/css"
     style="font:9pt/12pt times new roman;
            text-indent: 5mm;
            margin-left: 5mm">
WHEN in the Course of human Events, it
becomes necessary for one People to dissolve
the Political bands which have connected them
with another, and to assume among the Powers
of the Earth, the separate and equal Station
to which the Laws of Nature's God entitle
them, a decent Respect to the Opinions of
Mankind require they should declare the
causes which impel them to the Separation.
  <p type="text/css"
     style="font:9pt/12pt times new roman;
            text-indent: 5mm;
            margin-left: 5mm">
WE hold these Truths to be self-evident, that
all Men are created equal, that they are
endowed by their Creator with certain
unalienable Rights, that among these are
Life, Liberty, and the Pursuit of Happiness...
</body>
</html>
```

Fereastra browserului ar putea arăta astfel:



După cum observați valoarea atributului `STYLE` este destul de complicată. De fapt, sunt transmise mai multe atribute ale textului cum ar fi:
- dimensiunea fontului,
- distanța dintre două linii consecutive,
- culoarea,
- distanța față de una dintre margini etc.

O caracteristică a textului poate fi specificată cu ajutorul unei secvențe de tipul:
Denumire_caracteristică: valoare_caracteristică

Câteva dintre caracteristicile stilurilor sunt:
- `background`, `color`;
- `font-family`, `font-size`, `font-style`, `font-weight`;
- `line-height`;
- `margin-left`, `margin-right`, `margin-top`;
- `text-align`, `text-decoration` și `text-indent`.

Cu ajutorul caracteristicii `background` poate fi specificată o imagine sau o culoare de fundal. Valoarea transmisă poate fi URL-ul imaginii, denumirea culorii sau valoarea RGB a culorii.
Exemplu: `background: silver`

Prin intermediul caracteristicii `color` putem specifica o culoare pentru afișarea textului, valoarea care este transmisă putând fi denumirea sau valoarea RGB a culorii.
Exemplu: `color: white`

Caracteristica `font-family` ne permite să indicăm tipul fontului, ea putând avea ca valori un font sau o familie de fonturi.
Exemplu: `font-family: courier`

`Font-size` este folosit pentru a indica dimensiunea fontului, valoarea transmisă fiind un număr urmat de unitatea de măsură. Unitățile de măsură admise sunt:
- puncte (*pt*); - țoli (*in*);
- centimetri (*cm*); - pixeli(*px*).

Exemplu: `font-size:12 pt` - setează dimensiunea textului la 12 puncte tipografice.

Pentru a defini textele cursive vom folosi caracteristica `font-style` care poate lua valorile `normal` și `italic`.
Exemplu: `font-style: italic`

Grosimea fontului poate fi specificată cu ajutorul caracteristicii `font-weight` care poate avea ca valori:
extra-light, *light*, *demi-light*;
medium;
demi-bold, *bold* și *extra-bold*.
Exemplu: `font-weight: bold` - scrierea unui text folosind caractere îngroșate

Distanța dintre liniile de bază ale caracterelor este indicată de caracteristica `line-height`. Valorile pe care le putem transmite sunt compuse dintr-un număr, urmat de o unitate de măsură. Se acceptă o nouă unitate de măsură și anume procentul (*pc*).
Exemplu: `line-height:2 cm` - distanța dintre liniile de bază a două rânduri de text este de 2cm.

Caracteristicile `margin-left`, `margin-right`, respectiv `margin-top` indică distanțele până la marginea stângă, marginea dreaptă, respectiv marginea superioară a paginii. Valorile sunt aceleași ca pentru caracteristica `line-height`.
Exemplu: `margin-left: 15px.`

Pentru a specifica alinierea textului va trebui să folosim caracteristica `text-align` cu următoarele valori:
- `left` - aliniere la stânga;
- `right` - aliniere la dreapta;
- `center` - afișarea unui text centrat.

Exemplu: `text-align: center.`

Diferite modalități de evidențiere a textului sunt oferite de caracteristica `text-decoration` care poate lua valorile:
- `none` - nimic;
- `underline` - subliniere;
- `italic` - cursiv;
- `line-through` - tăiat.

Exemplu: `text-decoration: underline` - pentru a afișa un text subliniat

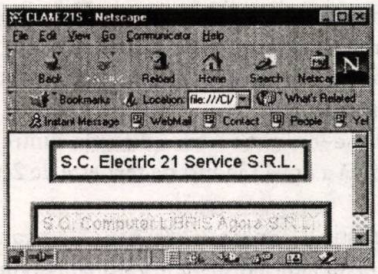
Ultima caracteristică pe care o vom prezenta este `text-indent`, ea putând fi folosită pentru indentarea primei linii a unui paragraf și acceptând aceleași valori ca și `line-height`.
Exemplu: `text-indent: 1 cm.`

După cum probabil ați observat, în exemplul anterior nu a fost respectată sintaxa care apare în specificațiile CSS. Aceasta se datorează faptului că browserele oferă anumite prescurtări pentru a micșora timpul necesar introducerii informațiilor. Astfel comenzile devin:

```
<p type="text/css"
   style="font-weight:bold;
          font-size:12pt;
          line-height:14pt;
          font-family:helvetica"> și
<p type="text/css"
   style= "font: bold 12pt/14pt helvetica">.
```

O altă posibilitate oferită de specificațiile CSS este setarea unor caracteristici pentru chenarele în care vor fi încadrate anumite elemente.

Exemplu:



Fișierul HTML care a dus la apariția acestei ferestre este următorul:

```
<html>
<head> <title> CLA&E21S </title> </head>
<body>
<h1 type=text/css
    style="text-align: center;
           font: 0.5cm helvetica;
           color: blue;
           border-width: 0.25cm;
           border-color: red;
           border-style: groove">
  S.C. Electric 21 Service S.R.L.
</h1>
<h1 type=text/css
    style="text-align: center;
           font: 0.5cm arial;
           color: aqua;
           border-width: 0.25cm;
           border-color: red;
           background: yellow;
           border-style: ridge">
  S.C. Computer LIBRIS Agora S.R.L.
</h1>
</body> </html>
```

Chenarul unui element poate fi afișat prin indicarea mărimii sale și a câtorva alte atribute. În exemplul anterior s-a stabilit dimensiunea de 0,25 cm, culoarea roșie și stilul groove. Valorile care pot fi transmise pentru caracteristica border-style sunt:

- none;
- dotted;
- dashed;
- solid;
- double;
- ridge;
- inset;
- outset.

Stiluri specificate în antet

Specificațiile HTML 4.0 oferă posibilitatea redefinirii efective a unor comenzi HTML prin intermediul tag-ului **<STYLE>** care apare în antetul documentului HTML. Să presupunem că doriți ca toate paragrafele să fie scrise cu fontul *Arial*, să aibă dimensiunea de 6 mm și să fie aliniate la centru. Pentru a realiza acest lucru, antetul documentului ar trebui să arate astfel:

```
<HEAD>
<STYLE type="text/css">
<!--
  P { font:6mm "Arial"; text-align: center}
-->
</STYLE>
</HEAD>
```

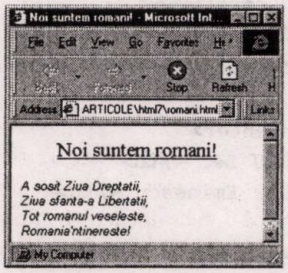
Din momentul în care ați creat această definiție, orice utilizare a marcajului **<H1>** va determina utilizarea stilului specificat în antet. La fel ca și în cazul script-urilor, definițiile de stiluri ar trebui înglobate în comentarii HTML, pentru ca browserele mai vechi, care nu recunosc modelele de stiluri, să nu interpreteze greșit comenzile date.

De asemenea, specificațiile CSS și HTML 4.0 permit crearea de noi tipuri de marcaje HTML. Să presupunem că doriți să dispuneți de două tipuri de paragrafe, unul pentru scrierea titlurilor și altul pentru scrierea strofelor unei poezii.

Exemplu

```
<html>
<head>
  <title> Noi suntem romani! </title>
  <style type=text/css>
  <!--
   P.titlu { font: 16pt;
             text-align: center;
             text-decoration: underline }
   P.strofa { font: 10pt arial;
              font-style: italic;
              text: left; }
  -->
  </style>
</head>
<body>
  <p class=titlu> Noi suntem romani!
  <p class=strofa>
  A sosit Ziua Dreptatii, <BR>
  Ziua sfanta-a Libertatii, <BR>
  Tot romanul veseleste, <BR>
  Romania'ntinereste!
</body>
</html>
```

Pagina browser-ului ar putea arăta astfel:



Cea mai importantă utilitate a acestei facilități o reprezintă posibilitatea de a schimba câteva caractere în antetul documentului. Pentru a modifica o întreagă pagină Web, fără a mai fi nevoie de modificări plictisitoare pentru fiecare element.

Fișiere .CSS

Specificarea tuturor stilurilor dorite la începutul documentului poate economisi timp, dar pentru mai multe pagini Web ar trebui scrise toate specificațiile la începutul fiecărui document.

Pentru a economisi timpul necesar pentru copierea acestor informații dintr-o pagină în alta, modelele de stiluri pot fi salvate în fișiere cu extensia `css`. Acest fișier va trebui încărcat în antetul fiecărei pagini care utilizează stilurile respective.

Exemplu

```
<html>
<head>
  <title> Fisier .CSS </title>
  <link rel=stylesheet
        href="stiluri.css"
        type=text.css>
</head>
<body>
 .......
</body>
</html>
```

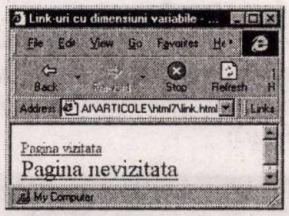
Link-uri cu dimensiuni variabile

Un artificiu interesant pe care îl puteți folosi pentru a îmbunătăți calitatea paginilor dumneavoastră WEB este acela de a modifica dimensiunile link-urilor în momentul în care acestea au fost vizitate.

Exemplu

```
<html>
<head>
  <title> Link-uri cu dimensiuni variabile
  </title>
  <style type=text/css>
  A: link {font:18 pt; color:blue}
  A: visited {font: 12 pt; color:blue}
  </style>
</head>
<body>
  <a href="pagina_vizitata.html">
    Pagina vizitata
  </a> <BR>
  <a href="pagina_nevizitata.html">
    Pagina nevizitata
  </a>
</body> </html>
```

La un moment dat, fereastra browserului ar putea arăta astfel:



Controlul amplasării elementelor în pagină

O altă facilitate oferită de specificațiile CSS o reprezintă posibilitatea de a realiza un control precis al amplasării elementelor în pagină, chiar de a suprapune sau a suprascrie unele elemente.

Exemplu

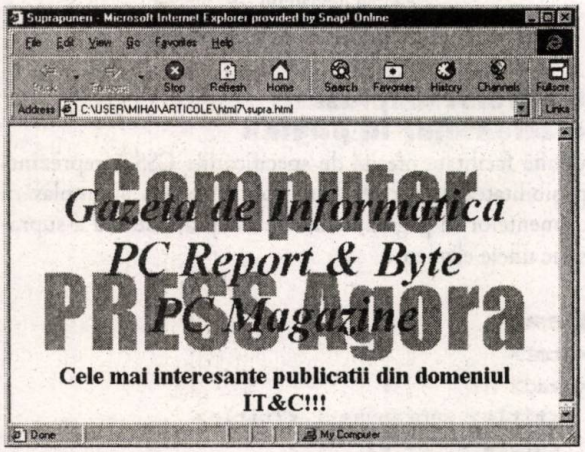
```
<html>
<head>
  <title> Suprapuneri </title>
  <style type=text/css>
   #background {color: #9999CC;
        font: 100px Impact, sans-serif;}
   #foreground {color: #893247;
       font: bold italic 50px Georgia, serif;}
  </style>
</head>
<body>
  <p align=center>
  <span id=background> Computer PRESS Agora
  </span>
  <p type=text/css
     style="font:bold 50px;
            margin-top: -200px;
            text-align: center;">
  <span id=foreground>
   Gazeta de Informatica<BR>
   PC Report &amp Byte<BR>
   PC Magazine
  </span>
  <p style="text-align: center;
           font: bold 25px serif;"
     type=text/css>
Cele mai interesante publicatii din domeniul
IT&ampC!!!
</body> </html>
```

Cel mai important atribut de stil utilizat în acest exemplu este `margin-top:-200px`, această caracteristică determinând ca marginea superioară a următorului text să fie mai "înaltă" cu 200 de pixeli decât în mod normal.

Este fructificat și un alt avantaj al unei metode de utilizare a modelelor de stiluri: utilizând atributul `ID` a fost creat un alt tip de specificație de model de stil care poate fi utilizată asemenea unui macrou asemănător celor din Microsoft Word. `ID`-urile sunt definite cu ajutorul prefixului `'#'` în cadrul comenzii **<STYLE>** și sunt folosite în cadrul documentului prin sintaxa:

```
ID=nume_identificator
```

Un alt tag folosit este **<SPAN>** care permite specificarea atributelor textului între opening tag-ul și closing tag-ul acestei comenzi. Pagina prezentată anterior este ilustrată în următoarea figură:



Din nou tabele

Specificațiile HTML 4.0 au introdus câteva comenzi noi pentru lucrul cu tabele.

Exemplu

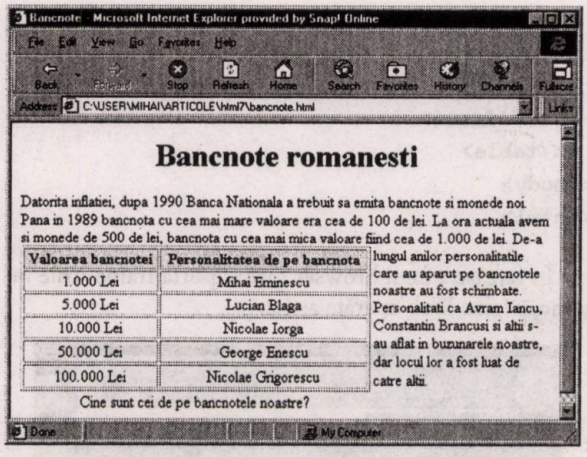
```
<html>
<head>
  <title>
    Bancnote
  </title>
</head>
<body>
 <h1 type=text/css
     style="text-align: center" >
  Bancnote romanesti
 </h1>
Datorita inflatiei, dupa 1990 Banca Nationala
a trebuit sa emita bancnote si monede noi.
Pana  in 1989 bancnota cu cea mai mare
valoare era cea de 100 de lei. La ora
actuala avem si monede de 500 de lei,
bancnota cu cea mai mica valoare fiind cea
de 1.000 de lei.
  <table border=1 width=66%
        align=left HSPACE=10 VSPACE=10>
 <tr bgcolor=#ffffcc#>
   <TH> Valoarea bancnotei </th>
   <TH> Personalitatea de pe bancnota </th>
 </tr>
 <tr align=center>
   <TD> 1.000 Lei </td>
   <TD> Mihai Eminescu </td>
 </tr>
 <tr align=center>
   <TD> 5.000 Lei </td>
   <TD> Lucian Blaga </td>
 </tr>
 <tr align=center>
   <TD> 10.000 Lei </td>
   <TD> Nicolae Iorga </td>
 </tr>
 <tr align=center>
   <TD> 50.000 Lei </td>
   <TD> George Enescu </td>
 </tr>
 <tr align=center>
   <TD> 100.000 Lei </td>
   <TD> Nicolae Grigorescu </td>
 </tr>
   <caption align=bottom>
   Cine sunt personalitatile de pe
   bancnotele noastre?
   </caption>
 </table>
De-a lungul anilor personalitatile care au
aparut pe bancnotele noastre au fost
schimbate. Personalitati ca Avram Iancu,
Constantin Brancusi si altii s-au aflat in
buzunarele noastre, dar locul lor a fost
luat cu timpul de alte personalitati.
</body>
</html>
```

În acest exemplu apare antetul de tabel care poate fi introdus cu ajutorul tag-ului **<TH>**. Această comandă are același rol ca și comanda **<TD>** dar, așa cum puteți vedea în imagine, textul este automat centrat și îngroșat. Din imagine se observă un alt efect interesant și anume faptul că textul apare și în jurul tabelului și nu numai deasupra și dedesubtul său.

Un alt element nou este tag-ul **<CAPTION>** care permite specificarea unui text explicativ asociat tabelului. Textul explicativ poate fi aliniat:

- în partea superioară a tabelului (`top`);
- în partea inferioară a tabelului (`bottom`);
- în stânga tabelului (`left`);
- în dreapta tabelului (`right`).

Iată și fereastra browserului pentru exemplul anterior:



Specificațiile HTML 4.0 introduc un nou atribut pentru comanda **<TD>** și anume **NOWRAP**. Dacă tag-ul **<TD>** primește acest parametru, browserul este forțat să nu despartă automat textul pe linii, decât dacă întâlnește comanda **
**.

O altă noutate introdusă de aceste specificații este valoarea `baseline` pentru atributul **VALIGN**. Această aliniere asigură ca prima linie a textului celulei să fie pe aceeași linie ca și cota inferioară a primei linii a textului din celula precedentă.

După cum probabil știți, un browser nu poate reda prima linie a tabelului până nu recepționează fiecare element de informație. Pentru a evita acest lucru este oferită posibilitatea specificării numărului de coloane a unui tabel cu ajutorul comenzilor **<COL>** și **<COLGROUP>**.

Exemplu:

```
<html>
<head>
  <title> Coloane </title>
</head>
<body>
  <table border=1>
  <colgroup>
    <col width=6"*">
    <col width=3"*">
    <col>
    <col width=25%>
    <col width=75>
  </colgroup>
  <tr> <TD> GInfo </td>
       <TD> GInfo </td>
       <TD> GInfo </td>
       <TD> GInfo </td>
  </tr>
  <tr> <TD> GInfo </td>
       <TD> GInfo </td>
       <TD> GInfo </td>
       <TD> GInfo </td>
       <TD> GInfo </td>
  </tr>
  <tr> <TD> GInfo </td>
       <TD> GInfo </td>
       <TD> GInfo </td>
  </tr>
  </table>
</body>
```

Să presupunem că lățimea ferestrei browserului este de 500 de pixeli. Am definit o coloană cu lățimea de 75 pixeli, una cu lățimea de 25% din lățimea totală a ferestrei, adică 0.25·500=125 de pixeli, iar restul de 300 pixeli vor fi împărțiți în proporția 6:3:1 între celelalte trei coloane, deci acestea vor avea 180, 90, respectiv 30 de pixeli lățime.

Comanda **<COL>** este echivalentă cu:

```
<col width=1"*">.
```

Specificațiile HTML 4.0 stabilesc faptul că într-un tabel se poate renunța la closing tag-uri pentru celule și linii. În plus au fost adăugate marcajele **<THEAD>** și **<TBODY>** pentru a indica antetul de tabel și corpul tabelului.

Un alt aspect interesant este posibilitatea alinierii conținutului unei celule relativ la un caracter, cu ajutorul unei comenzi de tipul:

```
<td align=char char=":">
```

Este posibilă și o altă opțiune pentru atributul `ALIGN` și anume alinierea de tip `justify`. De asemenea, prin specificarea unor atribute pentru o anumită coloană, acestea vor fi aplicate pentru toate celulele care fac parte din acea coloană.

Exemplu

```
<html>
<head>
  <title> Noi facilitati </title>
</head>
<body>
  <table border=1>
    <colgroup>
      <col align=justify>
      <col align=char char=":">
    </colgroup>
  <THEAD>
  <tr> <th> Activitate <th> Ora
  <TBODY>
    <tr> <td> Prima activitate   <td> 8:00 am
    <tr> <td> A doua activitate  <td> 8:30 am
    <tr> <td> A treia activitate <td> 9:45 am
    <tr> <td> ...                <td> ...
    <tr> <td> Ultima activitate  <td> 10:45 pm
  </table>
</body>
```

Din nefericire, Internet Explorer și Netscape Navigator nu oferă încă suportul complet pentru lucrul cu tabele, motiv pentru care nu avem posibilitatea să vă prezentăm

capturi de ferestre care să ilustreze aceste noi facilități. Totuși, firma Microsoft promite că versiunea Internet Explorer 5.0 care va apărea curând (probabil că a apărut deja în momentul în care citiți aceste rânduri) va oferi suport complet pentru toate facilitățile cuprinse în specificațiile HTML 4.0.

Din nou frame-uri

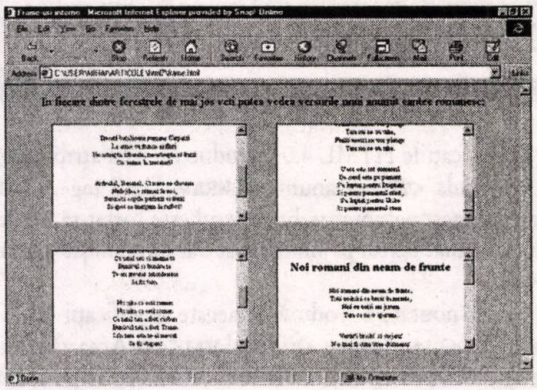
Ca în cazul tabelelor, specificațiile HTML 4.0 introduc câteva noutăți cu privire la frame-uri și anume conceptul de *frame intern* - frame complet înconjurat de o fereastră exterioară.

Un frame intern poate fi introdus într-o pagină Web cu ajutorul comenzii **<IFRAME>** care are o sintaxă asemănătoare comenzii **<FRAME>**. Între opening tag-ul și closing tag-ul acestei noi comenzi pot fi specificate comenzi ale căror efecte vor apărea în ferestrele browserelor care nu suportă frame-uri interne. Aceste comenzi vor fi ignorate de browserele care recunosc tag-ul **<IFRAME>**.

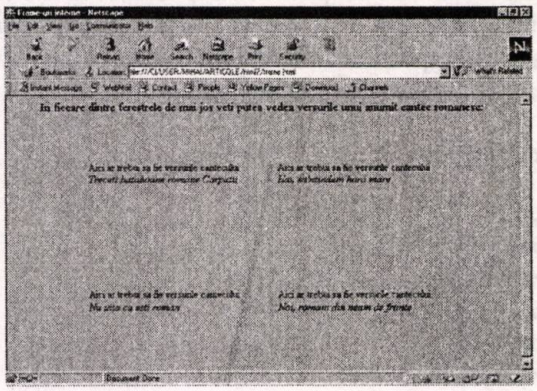
Exemplu

```
<html>
<head>
  <title>
    Frame-uri interne
  </title>
</head>
  <body bgcolor=#99cc99>
  <center>
  <font size=+1>
    In fiecare dintre ferestrele de mai jos
    veti putea vedea versurile unui anumit
    cantec romanesc:
  </font>
  </center>
  <table border=0 cellpadding=20
        align=center width=75% height=100%>
  <tr>
  <td>
    <iframe src="carpatii.html">
    Aici ar trebui sa fie versurile cantecului
    <i>Treceti batalioane romane Carpatii</i>
    </iframe>
  <v>
  <iframe src="horamare.html">
    Aici ar trebui sa fie versurile cantecului
    <i> Hai, sa'ntindem hora mare </i>
  </iframe>
  <tr>
    <TD>
    <iframe src="nuuita.html">
    Aici ar trebui sa fie versurile cantecului
    <i> Nu uita ca esti roman </i>
    </iframe>
    <td>
    <iframe src="romani.html">
    Aici ar trebui sa fie versurile cantecului
    <i> Noi, romani din neam de frunte </i>
    </iframe>
  </table>
</body>
</html>
```

În fereastra unui browser care suportă frame-urile interne, pagina Web ar arăta astfel:



În fereastra unui browser care nu suportă aceste frame-uri, aceeași pagină Web va arăta așa cum se vede în figura următoare:



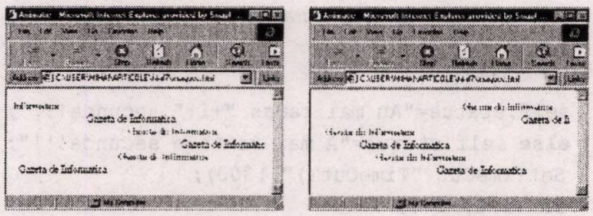
Comanda <MARQUEE>

Această comandă nu apare în specificațiile HTML 4.0, ea fiind introdusă de firma Microsoft și, ca urmare, browserul Internet Explorer este singurul care o suportă până în acest moment. Motivul pentru care am ales să prezentăm această comandă este faptul că poate oferi niște efecte interesante. Dacă veți folosi această comandă, textul care apare între opening tag-ul și closing tag-ul ei va fi animat. Unul dintre cei mai importanți parametri ai acestei comenzi este **BEHAVIOR** care poate lua valorile *scroll* (derulare), *slide* (alunecare) și *alternate* (alternat). Dacă valoarea este *scroll* atunci textul va apărea literă cu literă, va derula pe ecran, va dispărea și apoi va reapărea. În cazul valorii slide textul va aluneca pe ecran și se va opri în momentul în care va atinge marginea opusă. Dacă veți transmite valoarea alternate, textul se va balansa înainte și înapoi în ca-

drul zonei animate. Direcția în care va fi derulat textul poate fi specificată cu ajutorul parametrului **DIRECTION** care poate lua valorile left (stânga) și right (dreapta). Alți parametri ai acestei comenzi sunt **WIDTH, HEIGHT, BGCOLOR, VSPACE, HSPACE** etc., cu aceeași semnificație ca și la celelalte comenzi la care i-am mai întâlnit. Comanda dispune și de parametrii **SCROLLAMOUNT** și **SCROLLDELAY** care permit specificarea numărului de pixeli dintre două afișări succesive, respectiv intervalul de timp (în milisecunde) dintre două afișări. Iată și un exemplu de utilizare al acestui nou marcaj:

```
<html>
<head> <title> Animatie </title> </head>
<body>
  <marquee behavior=scroll direction=right>
    Gazeta de Informatica </marquee>
  <marquee behavior=scroll direction=left>
    Gazeta de Informatica </marquee>
  <marquee behavior=slide direction=right>
    Gazeta de Informatica </marquee>
  <marquee behavior=slide direction=left>
    Gazeta de Informatica </marquee>
 <marquee behavior=alternate direction=right>
    Gazeta de Informatica </marquee>
  <marquee behavior=alternate direction=left>
    Gazeta de Informatica </marquee>
</body> </html>
```

Imaginile următoare prezintă fereastra browserului în două momente diferite:



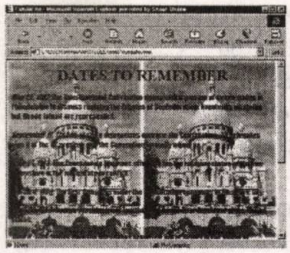
Fundaluri fixe

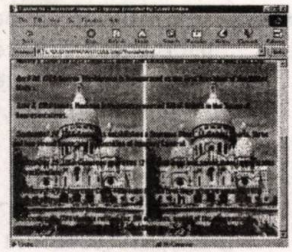
Dacă veți derula conținutul unei pagini Web atunci, de obicei, se mișcă și imaginea din fundal. Dacă doriți ca fundalul să rămână fix va trebui să transmiteți valoarea `fixed` pentru atributul **BGPROPERTIES** al comenzii **<BODY>**. Această comandă mai are doi parametri: **TOPMARGIN** și **LEFTMARGIN** care vă permit să specificați deplasamentul fundalului față de marginea din stânga, respectiv față de marginea superioară a paginii.

Exemplu

```
<html>
<head> <title> Fundal fix </title>
  <style type="text/css">
    #stil {font: 12pt Impact;}
  </style>
</head>
<body text="Nemes Liana"
      background="sacrecr.jpg"
      bgproperties=fixed>
  <h1 type=text/css style="text-align:center">
    DATES TO REMEMBER </h1>
  <span id=stil>
    <p> <i> May 25, 1787: </i>
    The Constitutional Convention opens with
    a quorum of seven states in Philadelphia
    to discuss revising the Articles of
    Confederation. Eventually all states but
    Rhode Island are represented.
    <p> <i> September 17, 1787: </i>
    All 12 state delegations approve the
    Constitution, 39 delegates sign it of
    the 42 present, and the Convention
    formally adjourns.
    <p> <i> June 21, 1788: </i>
    The Constitution becomes effective for
    the ratifying states when New Hampshire
    is the ninth state to ratify it.
    <p> <i> March 4, 1789: </i>
    The first Congress under the
    Constitution convenes in New York City.
    <p> <i> April 30, 1789: </i>
    George Washington is inaugurated as the
    first President of the United States.
    <p> <i> June 8, 1789: </i>
    James Madison introduces proposed Bill
    of Rights in the House of
    Representatives.
    <p> <i> September 24, 1789: </i>
    Congress establishes a Supreme Court, 13
    district courts, three ad hoc circuit
    curts, and the position of Attorney
    General.
    <p> <i> September 25, 1789: </i>
    Congress approves 12 amendments ans
    sends them to the states for
    ratification.
    <p> <i> February 2, 1790: </i>
    Supreme Court convenes for the first
    time after an unsuccessful attempt
    February 1.
    <p> <i> December 15, 1791: </i> Virginia
    ratifies the Bill of Rights, and 10 of
    the 12 proposed amendments become part
    of the U.S. Constitution.
    </span>
  </font>
</body>
</html>
```

Dacă priviți următoarele două imagini veți observa că fundalul rămâne fix, în timp ce conținutul paginii este derulat.





Dacă ați fost atenți la exemplul anterior ați observat că textul este scris cu o culoare a cărei denumire este cel puțin ciudată: `Nemes Liana`. (Orice asemănare cu persoane din lumea reală este mai mult sau mai puțin întâmplătoare.) În mod surprinzător această denumire pentru o culoare este acceptată de browsere, componentele RGB ale culorii fiind generate în funcție de codurile ASCII ale caracterelor transmise. Totuși, funcția după care se determină aceste componente RGB diferă de la un browser la altul, așa că nu este recomandabilă folosirea unei astfel de culori.

Despărțirea în silabe

Uneori, la sfârșitul unui rând rămâne un spațiu prea mare care ar putea fi supărător pentru vizitatorii paginilor dumneavoastră. Pentru a evita apariția acestor spații există posibilitatea specificării modului în care se vor despărți în silabe anumite cuvinte. Standardul HTML tratează secvența `` ca o indicație a sfârșitului unei silabe.

Dacă secvența apare într-un cuvânt care nu trebuie despărțit în silabe, atunci ea este ignorată, dar dacă respectivul cuvânt se află la sfârșitul rândului, atunci se va încerca afișarea câtorva silabe pe un rând, iar restul silabelor pe rândul următor. Versiunile actuale ale browserelor nu suportă încă despărțirea în silabe, ele înlocuind secvența `` cu o cratimă, chiar dacă nu este necesar.

Comanda <META>

Pentru ca paginile dumneavoastră WEB să fie cunoscute este necesar ca acestea să fie "prietenoase" față de motoarele de căutare. Există mai multe metode care vă permit acest lucru, dar cea mai importantă este utilizarea tag-ului **<meta>**.

Această comandă nu determină afișarea nici unei informații în fereastra browserului, dar toate motoarele de căutare vor folosi conținutul acesteia ca rezumat sau ca pe o scurtă descriere a paginii. Același tag permite specificarea unor cuvinte cheie ale paginii dumneavoastră, care vor fi luate în considerare de către motoarele de căutare.

Exemplu

```
<meta name=description
  contents="Vizitati pagina Web a Gazetei de
  Informatică pentru a deveni un viitor
  specialist in tehnologia informatiei!!!"
<meta name=keywords
  contents="GInfo, Gazeta de Informatica,
  HTML, Olimpiada de Informatica, Java,
  probleme propuse, probleme rezolvate,
  noutati, baze de date, begin"
```

Comanda **<meta>** poate fi utilizată pentru realizarea multor alte efecte spectaculoase, unul dintre cele mai interesante fiind posibilitatea de a schimba conținutul unei pagini Web după un anumit interval de timp.

De exemplu

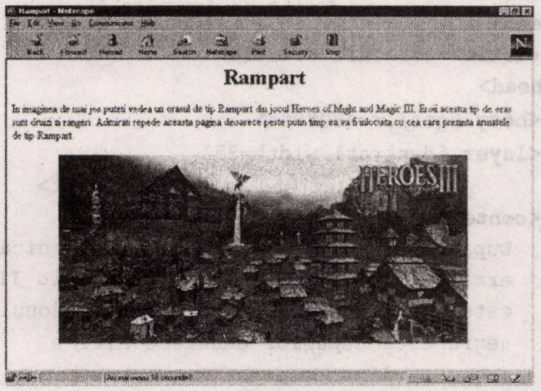
```
<meta http-equiv=refresh
      content="6;url=pagina2.html"
```

va duce la înlocuirea paginii curente cu pagina corespunzătoare fișierului `pagina2.html` după 6 secunde de la încărcarea paginii. Să considerăm următoarele două fișiere HTML:

```
<html>
<head>
  <title> Rampart </title>
  <meta name=description
    contents="Orasul de tip Rampart din
    Heroes of Might and Magic III">
  <meta name=keywords
    contents="Rampart, Heroes of Might and
    Magic III, Heroes III, Heroes 3, Heroes3">
  <meta http-equiv=refresh
      content="30;url=rampart_creatures.html">
  <script language="JavaScript">
  <!--
  i=30;
  function TimeOut(){
  if (i>=20)
  self.status="Au mai ramas "+i+" de secunde!!!";
  else
  if (i!=1)
  self.status="Au mai ramas "+i+" secunde!!!";
  else self.status="A mai ramas o secunda!!!";
  SetTimeOut("TimeOut()",1000);
  if (i>0) i--;
  }
  TimeOut();
  // -->
  </script>
</head>
<body>
  <h1 type=text/css
    style="text-align:center">
    Rampart
  </h1>
  In imaginea de mai jos puteti vedea un
  orasul de tip Rampart din jocul Heroes of
  Might and Magic III. Eroii acestui tip de
  oras sunt druizi si rangeri. Admirati
  repede aceasta pagina deoarece peste putin
  timp ea va fi inlocuita cu cea care
  prezinta armatele de tip Rampart.
  <p>
```

```
  <center>
    <img src="rampart.jpg" width=400>
  </center>
</body>
</html>

<html>
<head>
  <title> Armate Rampart </title>
  <meta name=description
    contents="Armatele de tip Rampart din
    Heroes of Might and Magic III">
  <meta name=keywords
    contents="Rampart, Heroes of Might and
    Magic III, Heroes III, Heroes3, Heroes
    III, centaur, centaur captain, dwarf,
    battle dwarf, wood elf, grand elf,
    pegasus, silver pegasus, dendroid guard,
    dendroid soldier, unicorn, war unicorn,
    green dragon, gold dragon">
</head>
<body bgcolor=#bfffbf>
  <table border=0>
  <tr>
  <TD> <img src="1centaur.gif"> <BR>
       <center> Centaur </center>
  <TD> <img src="1dwarf.gif"> <BR>
       <center> Dwarf </center>
  <TD> <img src="1woodelf.gif"> <BR>
       <center> Wood elf </center>
  <TD> <img src="1pegasus.gif"> <BR>
       <center> Pegasus </center>
  <TD> <img src="1dendroidguard.gif"> <BR>
      <center> Dendroid Guard </center>
  <tr>
  <TD> <img src="1centaurcaptain.gif"> <BR>
       <center> Centaur Captain </center>
  <TD> <img src="1battledwarf.gif"> <BR>
       <center> Battle Dwarf </center>
  <TD> <img src="1grandelf.gif"> <BR>
       <center> Grand Elf </center>
  <TD> <img src="1silverpegasus.gif"> <BR>
       <center> Silver Pegasus </center>
  <TD> <img src="1dendroidsoldier.gif"> <BR>
       <center> Dendroid Soldier </center>
  <tr>
  <TD> <img src="1unicorn.gif"> <BR>
       <center> Unicorn </center>
  <TD> <img src="1warunicorn.gif"> <BR>
       <center> War Unicorn </center>
  <td align=center VALIGN=bottom>
  <font size=5> Acestea sunt armatele de tip
  </font>
  <font size=6> Rampart</font>
  <TD> <img src="1greendragon.gif"> <BR>
       <center> Green Dragon </center>
  <TD> <img src="1golddragon.gif"> <BR>
       <center> Gold Dragon </center>
  </table>
</body>
</html>
```

Imediat după încărcarea primului fișier HTML, fereastra browserului ar putea arăta astfel:



După expirarea celor 30 de secunde pagina din fereastra browserului se va modifica, apărând următoarea pagină Web:

Pagini Web dinamice

Până în prezent, toate paginile Web pe care le-ați creat erau limitate la o singură dimensiune: totul se petrece pe o "coală de hârtie". Cu ajutorul modelelor de stiluri puteți controla poziționarea relativă a elementelor, le puteți chiar suprapune cu ajutorul unor artificii, dar rezultatele nu sunt foarte atractive.

Firma Netscape a introdus o nouă facilitate care oferă posibilitatea extinderii paginilor Web, astfel încât acestea să conțină niveluri (straturi) multiple. Pentru aceasta a fost introdusă o nouă comandă HTML și anume **<LAYER>**. Această comandă dispune de o gamă largă de parametri, dar cei mai importanți sunt **ID** și **WIDTH**.

Cel mai simplu strat poate fi definit în felul următor:

```
<layer id=strat width=300>
```

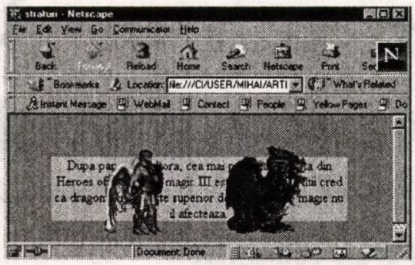
Cu ajutorul acestei comenzi este creat un strat denumit *strat*, de lățime 300 de pixeli.

Alte atribute ale acestui tag sunt: **BGCOLOR**, **BACKGROUND**, **TOP** și **LEFT**. Cu ajutorul primelor două atribute poate fi specificată o imagine fundal sau o culoare de fond pentru stratul respectiv. Atributele **TOP** și **LEFT** permit o localizare exactă a stratului relativ la colțul din stânga-sus.

Exemplu

```
<html>
<head>
  <title> straturi </title>
</head>
  <body bgcolor=#8080ff>
  <layer id=strat1 width=350
         top=50 left=50 bgcolor=#cfcfff>
  <center>
    Dupa parerea multora, cea mai puternica
    armata din Heroes of Might and magic III
    este arhangelul. Altii cred ca dragonul
    negru este superior deoarece nici o
    magie nu il afecteaza.
  </center>
  </layer>
  <layer id=strat2 width=350 top=25 left=75>
      <img src="1archangel.gif">
      <img src="1blackdragon.gif">
    </layer>
</body>
</html>
```

Efectul poate fi observat în imaginea următoare:



Caracteristica straturilor de *a putea fi plasate unul deasupra altuia* reprezintă un concept foarte important în dezvoltarea paginilor Web. Ca în cazul graficii pe calculator, straturile din paginile Web apar pe o axă verticală *Oz*, perpendiculară pe planul ecranului. Straturile care au valori mari pentru coordonatele *z* apar deasupra celor cu coordonate mai mici. Coordonatele *z* cresc pe măsură ce straturile sunt definite în pagină. De aceea un strat *strat1*, definit după un strat *strat2*, va apărea deasupra stratului *strat2*.

Există posibilitatea specificării valorilor coordonatei *z* a unui strat, în cazul în care doriți ca straturile să fie afișate într-o altă ordine. Pentru aceasta au fost introduși parametrii **Z-INDEX**, **ABOVE** și **BELOW** ai comenzii **<LAYER>**.

Cu ajutorul parametrului **Z-INDEX** puteți specifica direct coordonata verticală a stratului. Cu ajutorul celorlalți doi parametri puteți specifica faptul că un anumit strat se află imediat deasupra, respectiv imediat dedesubtul altui strat.

Nu puteți defini straturi cu aceeași coordonată verticală. Trebuie să aveți grijă să nu apară contradicții prin folosirea atributelor **ABOVE** și **BELOW**. Dacă nu respectați aceste condiții rezultatele nu sunt definite în specificațiile HTML și efectele diferă de la un browser la altul. De exemplu următoarea definiție de straturi nu ar trebui să apară în nici o pagină Web:

```
<layer id=strat1 above=strat2 width=100>
<layer id=strat2 above=strat1 width=100>
```

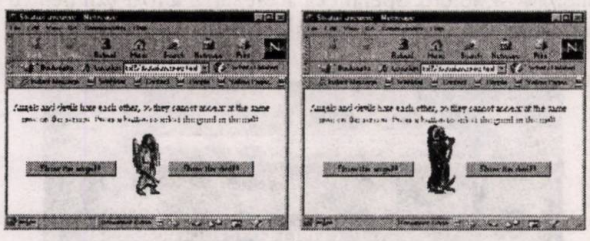
Starea de vizibilitate a unui strat este dată de atributul **VISIBILITY** care poate avea trei valori:
show (*afișat*) - stratul este vizibil;
hidden (*ascuns*) - stratul nu este vizibil;
inherit (*moștenește*) - stratul va avea aceeași stare ca și stratul părinte (există posibilitatea definirea de straturi imbricate).

Exemplu

```
<html>
<head>
  <title> Straturi ascunse </title>
</head>
<body>
  <layer id=strat2 width=50% top=25%
         left=25% visibility=show>
  <center>
    <img src="1angel.gif">
  </center>
  </layer>
  <layer id=strat3 width=50% top=25%
         left=25% visibility=hidden>
  <center>
    <img src="1devil.gif">
  </center>
  </layer>
  <layer id=strat1 top=10%>
  <center> Angels and devils hate each
    other, so they cannot appear at the same
    time on the screen. Press a button to
    select the good or the evil!
    <BR><BR><BR><BR>
    <form>
<input type=button value="Show the angel!!!"
       OnClick="strat2.visibility='show';
       strat3.visibility='hidden';
       return false;">
                
                
```

```
  <input type=button value="Show the devil!!!"
         OnClick="strat3.visibility='show';
         strat2.visibility='hidden';
         return false;">
  </form>
  </center>
  </layer>
</body>
</html>
```

Următoarele două imagini prezintă fereastra browserului în două situații: la încărcarea paginii și după apăsarea butonului *Show the devil!*. În funcție de butonul pe care apăsați va apărea una dintre cele două imagini.



Un alt efect interesant care poate fi obținut folosind o combinație între straturi și JavaScript este posibilitatea de a mișca o imagine în fereastra browserului. În exemplul următor va fi creat un strat care va conține imaginea unui titan care se va deplasa pe ecran. Pagina va mai include și două butoane *Start* și *Stop* cu ajutorul căruia puteți opri sau porni mișcarea titanului.

```
<html>
<head>
  <title> Titan </title>
  <script language=JavaScript>
  var dx=5;
  var dy=5;
  var miscare=1;
  function misca(){
    document.titan.top+=dx;
    if (document.titan.top>100) dx=-5;
                        //marginea inferioara
    if (document.titan.top<0) dx=5;
                        //marginea superioara
    document.titan.left+=dy;
    if (document.titan.left>400) dy=-5;
                       //marginea din dreapta
    if (document.titan.left<0) dy=5;
                        //marginea din stanga
    if (miscare==1) setTimeout(misca,10);
  }
  function Start(){miscare=1; misca();}
  function Stop(){miscare=0;}
  </script>
</head>
<body>
  <center>
  <h1 type=text/css
      style="font:32pt Impact">
  Titanul este cea mai puternica armata care
  are posibilitatea atacului de la distanta.
  </h1>
  <layer id=titan top=0>
    <img src="1titan.gif">
  </layer>
  <form>
    <input type=button value=Start
           OnClick="Start(); return false;">
                    
                    
    <input type=button value=Stop
           OnClick="Stop(); return false;">
  </form>
  </center>
</body>
</html>
```

Ideea de bază pentru construcția acestei pagini este simplă. La început, stratul care conține imaginea titanului se află în colțul din stânga-sus al ferestrei browserului, iar apoi locația sa este modificată prin adăugarea valorilor `dx`, respectiv `dy`, la coordonatele colțului stânga-sus al stratului. În cazul în care coordonata orizontală depășește o anumită valoare, atunci variabila corespunzătoare va primi o valoare negativă și imaginea se va deplasa, pe orizontală, de la dreapta la stânga.

Similar, în cazul în care coordonata orizontală va scădea sub o anumită valoare, atunci variabila corespunzătoare va primi o valoare pozitivă și imaginea se va deplasa de la stânga la dreapta. Am procedat asemănător pentru coordonata verticală.

Au fost definite trei funcții:

`Start()` - apelată pentru a pune în mișcare imaginea;

`Stop()` - apelată pentru a opri mișcarea acesteia;

`misca()` - modifică poziția imaginii în modul descris anterior, apoi verifică starea variabilei `miscare` și, dacă aceasta are valoarea 1, se autoapelează după 10 milisecunde.

Dacă această variabilă are valoarea 0, atunci execuția funcției `misca()` se termină și poziția stratului nu se mai modifică. Următoarele imagini prezintă starea paginii în două situații diferite:

Un efect interesant se obține dacă veți apăsa de mai multe ori pe butonul *Start*. Datorită faptului că la fiecare apăsare se va apela funcția `misca()`, imaginea se va mișca mai repede, deoarece pentru fiecare apel vor fi modificate coordonatele stratului.

Un alt parametru al tag-ului **<LAYER>** este **CLIP**. Cu ajutorul acestui parametru se definește regiunea de tăiere pentru strat (materialul corespunzător stratului poate fi mai mare decât mărimea curentă a acestuia; secțiunea mai mică din strat care va fi afișată reprezintă regiunea de tăiere). Pentru localizarea exactă a stratului față de fereastra browserului sunt puși la dispoziție alți doi parametri: **PAGEX** și **PAGEY**. Un strat poate conține o întreagă altă pagină HTML, URL-ul acesteia putând fi precizat cu ajutorul parametrului **SRC**.

Tot ce am spus până acum despre straturi este valabil numai pentru browserul Netscape Navigator. Pentru mai multe informații despre straturi în viziunea Netscape puteți vizita pagina:
http://developer.netscape.com/library/documentation/communicator/dynhtml/layers32.htm

Compania Microsoft implementează diferit paginile Web dinamice, felul în care browserul Internet Explorer le afișează fiind asemănător, dar comenzile HTML diferă foarte mult. Implementarea Microsoft este mai robustă, dar mult mai dificil de învățat. Un punct în plus pentru Microsoft este faptul că modelul său este compatibil cu modelul de document orientat pe obiecte (*Document Object Model - DOM*) care a fost propus de comitetul de standardizare HTML. Informații despre straturi în viziunea Microsoft puteți găsi la adresa:
http://www.microsoft.com/workshop/author/dhtml/

Contoare de pagini

Unul dintre cele mai interesante componente ale unei pagini Web este ***contorul*** care indică numărul vizitatorilor paginii. Contoarele pot fi realizate în multe feluri, ele pot fi de tip *grafic* sau de tip *text*.

Pentru *a crea un contor de tip text* trebuie să aflați dacă serverul care găzduiește pagina dumneavoastră oferă această facilitate. De aceea, modul în care sunt implementate contoarele diferă în funcție de tipul de server pe care sunt găzduite paginile.

De exemplu, pe anumite servere ar putea funcționa următoarea secvență de cod:

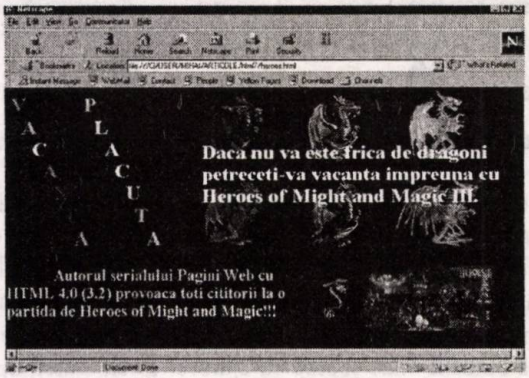
```
<font size=4>
  <b> Sunteti vizitatorul cu numarul
  </b>
</font>
<font size=5>
  <b> <#counter file="mycounter">
  </b>
</font>
```

Pentru un contor grafic trebuie să specificați o imagine specială care să reprezinte legătura cu un program de contorizare. O posibilitate ar fi **<img** `src="counter.cgi">`, fișierul `counter.cgi` trebuind să furnizeze imaginea corespunzătoare .

Pe Internet sunt disponibile o mulțime de contoare distractive și stiluri de afișare a numărului de vizitatori. O pagină Web de unde puteți afla tot ce doriți despre acestea este:http://www.digitmania.com

La final

Acesta a fost serialul despre limbajul HTML. Pentru informații referitoare la ultimele versiuni ale specificațiilor HTML vizitați pagina Web a consorțiului W3 la adresa http://www.w3.org. Sperăm că acest serial a fost util pentru cei care doresc să proiecteze site-uri Web și vă urăm:



Bibliografie:

1. Dave Taylor - *Crearea Paginilor Web cu HTML 4*, Editura Teora, 1998
2. Dick Oliver - *HTML 3.2 in 24 Hours*, 1997
3. Irina Athanasiu, Bogdan Costinescu, Octavian Andrei Drăgoi, Florentina Irina Popovici - *Limbajul Java. O perspectivă pragmatică*, Editura Computer Libris Agora, 1998
4. *Visual Basic, Programmer's Guide*, Microsoft Inc., 1997
5. *Visual Basic, Components Tools Guide*, Microsoft Inc., 1997
6. *PC Magazine România*, Ianuarie 1999
7. *Chip Computer Magazine*, Ianuarie 1999
8. *PC Report&Byte*, Februarie 1999
9. *România Liberă*, 10 Mai 1998
10. *Specificațiile HTML 4.0* - www.w3.org
11. *Heroes of Might and Magic III* - www.3do.com
12. *The Constitution of the United States and The Declaration of Independence*, National Constitution Center, 1997

Mihai Scorțaru este student la Universitatea Tehnică Cluj; poate fi contactat prin e-mail la adresa: scort@licj.soroscj.ro

Algoritmi ELEMENTARI

Sorin Groza

Pentru a putea rezolva cu succes o problemă de informatică este absolut necesar să cunoaștem câțiva algoritmi elementari. Vom prezenta în continuare probleme cu ajutorul cărora vom introduce algoritmi ce vor putea fi folosiți de către "pitici" în rezolvarea problemelor.

Concursul

La Clubul Copiilor se organizează un concurs la care participă n elevi, fiecare obținând un număr de puncte. Vor fi declarați câștigători toți cei cu punctaj maxim, fiecare primind câte un premiu.

Scrieți un program care va determina punctajul maxim și va comunica organizatorilor câte premii să pregătească.

Date de intrare:

```
n=11
16 72 9 7 72 41 12 16 72 16 16
```

Date de ieșire:

```
Se acorda 3 premii
Punctaj maxim: 72 puncte
```

Pentru rezolvarea aceastei probleme va trebui să găsim cel mai mare punctaj și să determinăm numărul de apariții al acestuia în șir. Aici apare unul din algoritmii cel mai des folosiți: determinarea ***maximului dintr-un șir de numere întregi***. Vom considera că cel mai mare element al șirului este cel de pe prima poziție și îl vom păstra într-o variabilă (max). Vom lua, unul câte unul, fiecare element al șirului și le vom compara cu max. În cazul în care un element este mai mare decât max, vom păstra în max valoarea acestuia. După parcurgerea întregului șir, în variabila max vom avea cel mai mare element al șirului. Numărul de premii îl vom calcula cu ajutorul variabilei nr_premii; vom parcurge șirul și, în momentul în care un element este egal cu max, vom incrementa acestă variabilă.

Listing: CONCURS.PAS

```
var i,max,n,nr_premii:Integer;
    p:array[1..100] of Integer;
Begin
  Write('Introduceti nr. concurentilor: ');
  Readln(n)
  Writeln('Punctele concurentilor:');
  for i:=1 to n do Read(p[i]);
  max:=p[1];{ determinarea maximului sirului }
  for i:=1 to n do
    if p[i]>max then max:=p[i];
                    { sfarsit determinare }
  for i:=1 to n do
    if p[i]=max then Inc(nr_premii);
  Writeln('Se acorda ',nr_premii,' premii.');
  Writeln('Punctajul maxim ',max,' puncte.')
End.
```

Grupuri

În curte se joacă n copii. Ei doresc să formeze grupuri a câte trei copii, astfel încât fiecare grup să aibă proprietatea: *înălțimea copilului din mijloc să fie media aritmetică a înălțimilor celorlalți doi copii.* Scrieți un program care să afișeze toate grupurile care se pot forma.

Date de intrare:

```
n=6
150 155 154 160 152 156
```

Date de ieșire:

```
150 152 154
152 154 156
150 155 160
154 155 156
152 156 160
```

Pentru a rezolva problema vom genera toate tripletele (*prim - mijloc - ultim*) și vom verifica dacă respectă condiția cerută. Deoarece, la un moment dat, un grup identificat printr-un triplet nu are valorile ordonate crescător și astfel verificarea proprietății cerute ar fi greoaie, vom ordona crescător înălțimile copiilor. Astfel, fiecare grup format are cele trei valori în ordine crescătoare. Acest algoritm se numește ***ordonarea crescătoare a unui șir***. Parcurgând șirul vom verifica dacă perechile de elemente vecine au valorile ordonate crescător. În caz contrar vom schimba între ele aceste valori. Vom face atâtea treceri până când nu mai au loc astfel de schimbări, moment în care șirul este ordonat crescător.

Listing: GRUPURI.PAS

```
var i,n,prim,mijloc,ultim,aux:Integer;
    h:array[1..100] of Integer;
    gata,se_poate:Boolean;
Begin
  Write('Introduceti numarul copiilor ');
  Readln(n);
  Writeln('Inaltimile copiilor:');
  for i:=1 to n do Read(h[i]);
  repeat  { ordonarea crescatoare a sirului }
    gata:=true;          { metoda "bulelor" }
    for i:=1 to n-1 do
      if h[i]>h[i+1]
      then
      begin
        aux:=h[i];
        h[i]:=h[i+1];
        h[i+1]:=aux;
        gata:=false
      end;
  until gata;            { sfarsit ordonare }
  for prim:=1 to n-2 do
    for mijloc:=prim+1 to n-1 do
      for ultim:=mijloc+1 to n do
        if h[mijloc]-h[prim]=
           h[ultim]-h[mijloc]
        then
        begin
          Write(h[prim],' ',h[mijloc],' ');
          Writeln(h[ultim]);
          se_poate:=true
        end;
  if not se_poate then
   Writeln('Nu se pot forma astfel de grupuri')
End.
```

Hoții

Un număr de `n` hoți ajung la un tezaur care constă din `m` seifuri, în fiecare seif existând un anumit număr de bani. Hoții vor lua bani dintr-un număr de seifuri consecutive ce respectă proprietatea: *suma banilor din aceste seifuri să se împartă exact la cei `n` hoți (fiecare hoț să "câștige" aceeași sumă) și să fie cea mai mare posibilă de câștigat.*

Spuneți care dintre seifuri le vor deschide hoții. Dacă problema nu are soluție, se va afișa mesajul `Ne lasam de meserie`.

Date de intrare:

```
n=4
m=9
2 10 30 3 6 6 6 6 2
```

Date de ieșire:

```
Deschid seifurile: 2 3
```

Pentru această problemă vom prezenta doi algoritmi având complexități diferite, adică pentru același set de date timpul de execuție diferă. Rulați aceste două programe cu date identice și observați timpul de execuție!

Primul algoritm generează toate subșirurile cuprinse între două poziții `i` și `j`, pentru fiecare subșir calculându-se suma elementelor. Dacă această sumă se divide la numărul de hoți și este cea mai mare până în acest moment, se păstrează această secvență, adică poziția de început, poziția de sfârșit și suma elementelor ei. Acest algoritm nu este optim, deoarece face unele calcule de mai multe ori, ceea ce determină creșterea timpului de execuție.

Listing: HOTII_1.PAS

```
var a:array[1..100] of Byte;
    i,j,x,n,s,nrhoti,max,poz1,poz2:Byte;
Begin
  Write('Introduceti numarul hotilor: ');
  Readln(nrhoti);
  Write('Introduceti numarul de seifuri: ');
  Readln(n);
  Writeln('Introduceti sumele din seifuri:');
  for i:=1 to n do Read(a[i]);
  for i:=1 to n-1 do   { generare subsiruri }
  begin
    for j:=i+1 to n do
    begin
      for x:=i to j do s:=s+a[x];
      if s mod nrhoti=0
      then
        if s>max then
        begin max:=s;
              poz1:=i; poz2:=j
        end;
      s:=0
    end
  end;                          { sfarsit generare }
  if max=0 then
  begin
    Writeln('Ne lasam de meserie'); Halt
  end;
  Write('Deschid seifele:');
  for i:=poz1 to poz2 do Write(i,' ')
End.
```

Al doilea algoritm utilizează un șir care reduce substanțial numărul calculelor și implicit timpul de execuție. Acest șir se numește ***șirul sumelor parțiale***. Astfel, elementul `sp[i]` va fi egal cu suma tuturor elementelor șirului `a` de la început până în a `i`-a poziție:

`sp[i]=a[1]+a[2]+...+a[i].`

Deoarece:

`sp[i]=a[1]+...a[i-1]+a[i]=sp[i-1]+a[i],`

se observă relația de calcul a elementelor acestui șir:

`sp[i]=sp[i-1]+a[i].`

Pentru a determina suma valorilor seifurilor între două poziții `i` și `j`, nu mai este necesară însumarea `a[i]+...+a[j]`, ci doar o simplă diferență `sp[j]-sp[i-1]`.

Un exemplu concret ar fi următorul: dacă vrem să calculăm suma valorilor seifurilor de la 5 la 9, vom determina diferența dintre sp[9] și sp[4], adică vom scădea din suma primelor 9 seifuri suma primelor 4 seifuri, obținând suma seifurilor de la 5 la 9. Șirul sumelor parțiale poate fi utilizat în rezolvarea multor probleme. Dacă o astfel de secvență are suma mai mare și se divide la numărul hoților, se păstrează poziția de început, poziția de sfârșit și suma secvenței. În final se vor tipări indicii seifurilor ce constituie secvența sau, mesajul cerut în cazul în care nu există o astfel de secvență.

Listing HOTII 2.PAS

```
var a:array[1..100] of Byte;
    sp:array[0..100] of Word;
    i,n,nrhoti,max,poz1,poz2,v,p1,p2:Byte;
Begin
  Write('Introduceti numarul hotilor: ');
  Readln(nrhoti);
  Write('Introduceti numarul de seifuri: ');
  Readln(n);
  Writeln('Introduceti sumele seifurilor:');
  for i:=1 to n do Read(a[i]);
  for i:=1 to n do
    sp[i]:=sp[i-1]+a[i];
          { creare sir sume partiale      }
          { determinare subsir de seifuri }
  for p1:=0 to n-1 do
    for p2:=p1+1 to n do
    begin
      v:=sp[p2]-sp[p1];
      if (v>max) and (v mod nrhoti=0)
      then
      begin
        max:=sp[p2]-sp[p1];
        poz1:=p1; poz2:=p2
      end
    end;                  { sfarsit determinare }
  if max=0 then
  begin
    Writeln('Ne lasam de meserie '); Halt
  end;
  Write('Deschid seifele: ');
  for i:=poz1+1 to poz2 do Write(i,' ')
End.
```

În încheiere vă propunem să rezolvați câteva probleme. Încercați să scrieți algoritmi diferiți pentru aceeași problemă și să estimați care sunt mai performanți.

Expoziția

La expoziția auto de la New York sunt prezentate n mașini, pentru fiecare dintre ele cunoscându-se prețul de vânzare. Scrieți un program care calculează diferența maximă de preț dintre două mașini prezentate.

Date de intrare:

```
n = 5
180 66 90 20 60
```

Date de ieșire:

```
160
```

Vizita

Într-un sat este agitație, deoarece sosește în vizită o mare personalitate, iar persoana cea mai în vârstă din sat trebuie să o întâmpine cu pâine și sare. Deoarece nu se știe cine este această persoană, este necesar ca toți locuitorii satului să-și declare data nașterii (*ziua, luna, anul*). Scrieți un program care, plecând de la aceste date, afișează pe ecran data nașterii persoanei celei mai în vârstă din sat.

Date de intrare:

```
n = 5
10      11      1915
22      12      1934
 9      1       1955
18      8       1915
25      11      1960
```

Date de ieșire:

```
persoana cea mai in varsta este nascuta in 18
8 1915
```

Competiția

La un concurs de tras cu frânghia, desfășurat pe plaja din Malibu, participă n persoane, fiecare dintre ele fiind caracterizată de un număr ce reprezintă *puterea* acesteia. Primii p (p≤n) participanți aparțin echipei STANGA, iar ceilalți începând cu poziția p+1, până la ultimul formează echipa DREAPTA.

Se va considera câștigătoare echipa care are suma puterilor participanților cea mai mare

a) citind o poziție p să se spună care echipă va fi învingătoare.

b) să se găsească o poziție astfel încât cele două echipe formate să lupte la infinit, adică să aibă aceeași sumă a puterilor. Dacă nu este posibil, să se afișeze mesajul IMPOSIBIL.

Date de intrare:

```
n=7
8 1 3 2 5 3 2
p=4
```

Date de ieșire:

```
STANGA
3
```

Domnul Sorin Groza este profesor la Clubul Copiilor din Aiud și poate fi contactat prin intermediul redacției.

Vă place GInfo?

Nume: _____ Prenume: _____ Vârsta: _____

Adresa: _____

Localitate: _____ Cod poştal: _____ Judeţ: _____ Telefon: _____

Ocupaţia: elev(ă) informatică ☐ elev(ă) ☐ student(ă) ☐ cadru didactic ☐ altă ocupaţie ☐ (precizaţi:_____)

Şcoala (firma, organizaţia etc.): _____

Dotări

Folosiţi un calculator acasă? Da ☐ Nu ☐

Câte calculatoare sunt instalate în şcoală: (aprox.) _____

Aveţi acces la e-mail: Da ☐ Nu ☐

Aveţi acces la WWW: Da ☐ Nu ☐

Cum apreciaţi dotarea cu tehnică de calcul a şcolii: _____

Aprecieri generale

Acordaţi note de la 1 la 10 pe următoarele criterii:

Structura tematică: ___

Calitatea articolelor: ___

Prezentare grafică: ___

Calitate tipografică: ___

Consideraţi că preţul este:

prea mare ☐ accesibil ☐ prea mic ☐

De cât timp cunoaşteţi Gazeta de Informatică (ani)? _____

Cum aţi procurat acest număr:

chioşc ☐ abonament ☐ cadou ☐ altfel ☐

În acest număr

Apreciaţi cu note de la 1 la 10 articolele acestui număr:

Reţele neuronale ___

Modelul relaţional ___

Informaţiile din memoria CMOS ___

Criptografia în sisteme electronice de plăţi ___

De la FORTRAN la JAVA ___

Easter Eggs ___

Pagini Web cu HTML 4.0 ___

Algoritmi elementari ___

Articolul care v-a plăcut cel mai mult: _____

De ce? _____

Articolul care v-a plăcut cel mai puţin: _____

De ce? _____

Consideraţi că ponderea problemelor rezolvate este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că ponderea problemelor propuse este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că ponderea subiectelor teoretice este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că nivelul de accesibilitate al articolelor este

prea coborât ☐ potrivit ☐ prea ridicat ☐

În viitor

În următoarele numere aşteptaţi:

mai multe articole de programare? Da ☐ Nu ☐

mai multe articole teoretice? Da ☐ Nu ☐

mai multe articole despre Internet? Da ☐ Nu ☐

mai multe prezentări de programe? Da ☐ Nu ☐

prezentări de jocuri? Da ☐ Nu ☐

articole despre hardware? Da ☐ Nu ☐

V-ar plăcea să citiţi:

pagini redactate de elevii unor licee? Da ☐ Nu ☐

lucrări premiate, realizate de elevi? Da ☐ Nu ☐

actualităţi şcolare? Da ☐ Nu ☐

prezentarea unor situri Web? Da ☐ Nu ☐

tips & tricks? Da ☐ Nu ☐

Comentarii:

Dacă spaţiul nu vă este suficient, folosiţi coli albe suplimentare

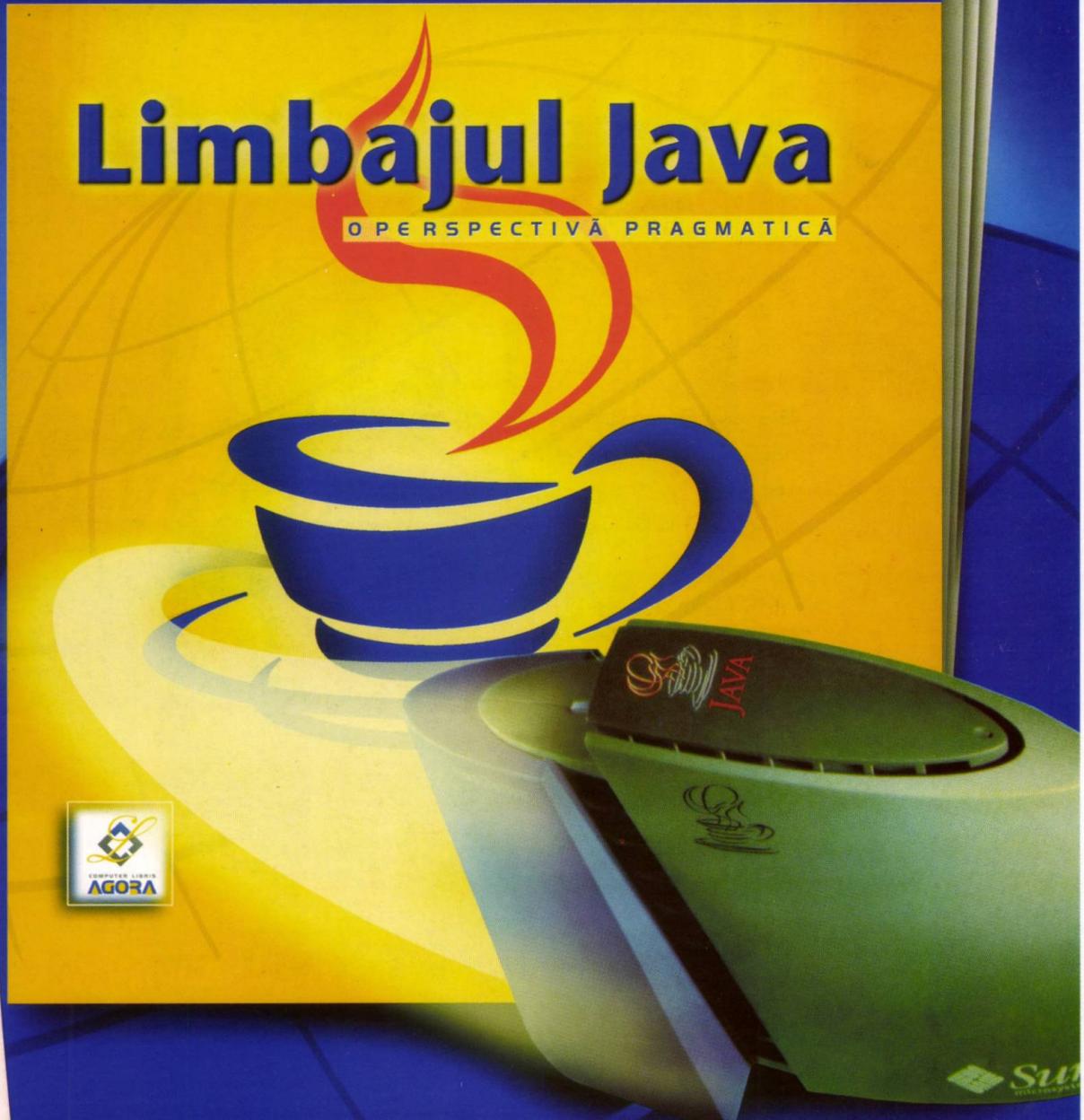
artea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Cartea AGORA • Car

COMPUTER LIBRIS AGORA

Irina Athanasiu
Octavian Andrei Drăgoi
Bogdan Costinescu
Florentina Irina Popovici

Limbajul Java

O PERSPECTIVĂ PRAGMATICĂ



ER LIBRIS AGORA Cluj-Napoca, Str.Universității Nr.8/8, 3400, Tel. 064-192422
ER PRESS AGORA Tg.Mureș Str.Tudor Vladimirescu Nr. 63/1 (CP 172/1), Tel. 065-166516, Fax 065-166290
Bucureşti: Str.Poterași, Nr.35/2 (CP 94/49), Tel: 01-3309281, Fax: 01-3309285

Un sfat prinde bine...

PC MAGAZINE ROMÂNIA

...oricând!

Dacă vrei să te orientezi printre miile de produse și tehnologii informatice de ultimă oră

ABONEAZĂ-TE!

Câștigi **30 000** lei și vei primi cadou și CD-urile **WINSTONE '99** și **WINBENCH '99**

COMPUTER PRESS AGORA, Str.Tudor Vladimirescu Nr.63/1, 4300 Tg.Mureş, Tel.065-166516, Fax 065-166290

POȘTALĂ PO
RĂSPUNS CA
POȘTALĂ PO

Octombrie 1999
Vol. 9/6

Lei 15.000

Gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:

PRINCIPIUL INDUCȚIEI ÎN INFORMATICĂ

CONCURSURI

Principiul cutiei lui Dirichlet

Pagini WEB cu JavaScript

Memorii cache

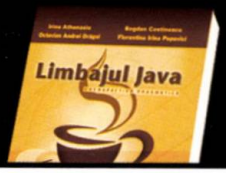
Programe virus și antivirus

GAZETA DE INFORMATICĂ

Bursele Agora!

Concurs pentru elevi organizat de Gazeta de Informatică

Runda 1



COMPUTER LIBRIS AGORA

COLARE

MANUAL Clasa a IX -a

Manualul este astfel conceput încât să permită învățarea noțiunilor prezentate cu ușurință, realizarea unor aplicații utile, individual sau în grupuri de elevi. Sunt incluse multe probleme rezolvate și propuse din care profesorul poate să aleagă, în funcție de nivelul general al clasei și importanța și dificultatea problemelor. Problemele mai dificile de la sfârșitul fiecărui capitol ar putea fi date elevilor ca "miniproiecte" de aplicații în scopul deprinderii lor cu munca în echipă. Autorii acestui manual au căutat să ofere un instrument didactic prin utilizarea căruia să crească eficiența procesului de predare și învățare și să împartă propria lor experiență didactică, acumulată pe parcursul a peste 25 de ani.

Limbaj

Algo

fundam

Clasa a X -a

Manualul prezintă tehnici și metode de programare ca: recursivitate, alocare dinamică, metodele backtracking, programare dinamică, greedy, divide et impera și branch and bound. Prezentarea este accesibilă, iar aplicațiile sunt diverse: grafică (fractali), jocuri, teoria grafurilor.
Manualul a fost conceput, plecând de la ediția 1993 (Limbajul Turbo Pascal, clasa a X-a) care a fost regândit substanțial și îmbogățit, astfel încât să fie accesibil unui elev de clasa a X-a. Ediția anterioară s-a bucurat de un succes deosebit, fiind apreciată de mii de cititori.

Două noi apariții

Pentru comenzi și informații suplimentare ne puteți contacta la tel. 064-192422

COMPUTER LIBRIS AGORA

Vol. 9, Nr. 6/1999

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Au mai colaborat:
Sorin Groza (Aiud)
Mihai Oltean (Cugir)
Angela Pop (Reșița)
Doru Popescu Anastasiu (Slatina)
Florin Sabău (Reșița)
Andrei Vancea (Cluj)
Lucian Vințan (Sibiu)

Tehnoredactare:
Dan Crețu, Călin Borz
Coperta și grafica:
Mircea Drăgoi

Adresa pentru corespondență:
C.P. 172-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: (065) 166.516
Fax: (065) 166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *Master Druck*
Târgu-Mureș
tel: 065/162.359
Editor: Computer Press Agora srl
str. T. Vladimirescu nr. 63/1
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Gabriela Bucșa (*gbucsa@agora.ro*)
Laura Dumitru (*ldumitru@agora.ro*)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: (01) 330.92.81

Sumar

probleme.. PAG. 4

propuse pentru REZOLVARE

Tabăra de selecție a concurenților pentru CEOI '99 și BOI '99

soluții.. PAG. 10

probleme REZOLVATE

Soluțiile unor probleme propuse la ONI '99

begin.. PAG. 18

Algoritmi de "VACANȚĂ"
Sorin GROZA

Probleme pentru începători, respectiv elevi de gimnaziu

tehno.. PAG. 21

Memorii CACHE
Lucian VINȚAN

babel.. PAG. 46

Programe virus și antivirus
Angela POP

concurs.. PAG. 26

RUNDA 1
Bursele AGORA!
Concurs pentru elevi, organizat de Gazeta de Informatică

actual.. PAG. 28

I. Concursuri
II. Hardware & Software
... și alte știri

focus.. PAG. 30

Princpiul inducției în informatică
Doru POPESCU ANASTASIU

Principiul cutiei lui Dirichlet
Mihai OLTEAN

serial.. PAG. 37

Pagini WEB cu JavaScript
Mihai SCORȚARU

Din 25 octombrie 1999
Preț Super Special pentru abonați
449$
Calculator Alpis (fără monitor, toate taxele plătite)
BRINEL

BRINEL Business Partners:
Cluj-Napoca: BRINEL, Blvd. N. Titulescu #4, tel: 064-414.610
* Alba-Iulia: Info DSC SRL, 058-827.144
* Baia Mare: Multinet SRL, 062-222.201
* București: BRINEL, 094-547.804
* Brașov: Cosidor Computer, 068-310.534
* Dej: Eustel Prod Com, 064-433.713
* Râmnicu Vâlcea: Cibpro SRL, 050-734.894
* Sibiu: Alfasoft SA, Sucursala, 069-213.962
* Suceava: Globe Seattle PC, 030-215.497
* Târgu Mureș: Electro-Orizont SRL, 065-168.807
* Timișoara: AS Computer,

Misiune: Inteligența globală
Clasificare: Prioritate maximă
Nume de Cod: DevDays '99

statistică globală — **64 țări - 159 orașe - peste 70,000 dezvoltatori**

Africa de Sud	Canada	Danemarca	Franța	Israel	Noua Zeelandă	Porto Rico	Suedia
Arabia Saudită	Chile	Rep. Dominicană	Germania	Italia	Norvegia	România	Taiwan
Argentina	China	Ecuador	Grecia	Iugoslavia	Olanda	Rusia	Tailanda
Australia	Columbia	Egipt	Guatemala	Japonia	Panama	Scoția	Turcia
Austria	Coreea	Elveția	Hong Kong	Macedonia	Paraguay	Singapore	Statele Unite
Belgia	Costa Rica	Emiratele Arabe Unite	India	Malaezia	Peru	Slovacia	Uruguay
Bolivia	Croația	Finlanda	Indonezia	Marea Britanie	Polonia	Slovenia	Ungaria
Brazilia	Rep. Cehă	Filipine	Irlanda	Mexic	Portugalia	Spania	Venezuela

Scopul acestei misiuni este transferul de cunoștințe.

Ca membru de bază al comunității dezvoltatorilor de aplicații din România, sunteți invitat să luați parte la cel mai important eveniment al anului pentru dezvoltatorii de aplicații: Developer Days '99 - ediția a II-a

Sesiunea de anul acesta se referă la utilizarea tehnologiei Windows DNA în aplicațiile pentru platforma Windows 2000.

Developer Days '99, București, 18 noiembrie 1999,

Centrul de conferințe Romexpo, Sala Nicolae Titulescu, orele 09:00 - 17:00

Microsoft® Developer Days

OFERIT DUMNEAVOASTRĂ DE msdn

Înregistrare, conținut sesiune și prezentatori:

http://www.onlineservices.ro/devdays

Înregistrați-vă on line pe site-ul de mai sus sau la Microsoft Infocenter, tel.: 410 92 17

Microsoft®

Pentru SUCCESUL DEPLIN

Un nou an școlar, o nouă programă totdeauna îi pune la încercare pe profesori și pe elevi. Dar conținutul nu s-a schimbat doar de dragul schimbării. Evident, trebuie găsite metodele cele mai potrivite care să garanteze atingerea obiectivelor noi, specifice timpului în care trăim și disciplinei pe care o predăm/învățăm.

Munca în echipă

Studiind programe școlare din țările lumii, precum și metode de predare recomandate în universități și școli, putem observa că în ultimii ani tot mai mare accent se pune pe dezvoltarea abilităților necesare ***muncii în echipă***. Grija manifestată în învățământ față de acest obiectiv are o explicație multiplă care ar trebui să ne pună pe gânduri.

În primul rând aș vrea să atrag atenția asupra unui aspect care unora s-ar putea să li se pară curios: promovarea muncii în echipă este susținută în mod accentuat în cazul disciplinelor care folosesc ca ***instrument de bază calculatorul:*** *Tehnologia informației, Informatica, Algoritmi și programare* etc. Explicația vine în mod firesc dacă observăm că în timpul lucrului cu calculatorul, elevul - viitorul utilizator, sau programator - de regulă lucrează *singur*. Dar lucrând singur se izolează (vrând-nevrând) de colegii săi, se scufundă într-un dialog pe care îl poartă doar cu mașina. Dacă nu există posibilitatea de a comunica pe parcursul învățării/muncii cu ceilalți, această mașină poate să devină "prieten" sau "dușman", oricum tot mașină rămâne. Este nevoie stringentă de a păstra legătura cu ceilalți copii, de a vorbi cu ei și de a-i încuraja să colaboreze pe parcursul pregătirii unei aplicații, altfel elevul, în singurătatea sa, fie va crede că este în stare să facă lucruri unice pe care ceilalți nu le pot realiza, fie își va pierde curajul și se va simți "subjugat" și depășit de tehnologie.

Așa cum în echipele de proiectanți, dacă nu zilnic, săptămânal au loc întruniri la care membrii echipei își prezintă ultimele rezultate, au loc dezbateri, se fac propuneri, aprecieri, ar fi foarte benefic, dacă în școli s-ar proceda asemănător. Dar ce să facă bietul elev, mai ales cel de la profilul informatică, în condițiile în care profesorul, după ce a intrat în laborator și a enunțat o problemă, nu face altceva decât să se plimbe printre calculatoare și așteaptă ca cineva să termine aplicația? Dacă problema nu este analizată în detaliu de elevii care urmează să încerce să o rezolve, dacă nu se ține cont de părerea fiecăruia, dacă după finalizare nu se dezbat posibilele variante, practic nu s-a făcut nimic. Este posibil ca unii copii să rezolve problema corect, dar dacă nu li se cere să vorbească despre felul în care au gândit-o, s-ar putea ca procesul intelectual respectiv să se volatilizeze, și din punct de vedere instructiv-educativ, acest profesor să nu poată demonstra eficiența muncii sale. De asemenea, este posibil ca unii copii să nu fie în stare să rezolve singuri problema: vor "trage cu ochiul", vor întreba în șoaptă câte ceva de la colegul mai "deștept", astfel încărcându-se cu sentimentul vinovăției. Să nici nu mai luăm în considerare cazul în care profesorul nostru ar "vâna" asemenea "colaborări" cu scopul de a-l penaliza pe "vinovat".

Ideal ar fi să se formeze echipe de elevi, să li se împartă sarcini, să existe responsabilități bineprecizate. Un astfel de stil de lucru se poate introduce de la primele lecții: unul cu algoritmul, celălalt cu implementarea, unul cu notițele, altul cu desenele etc.

Exprimarea

Vă propun următorul experiment: cereți colegului, copilului sau elevului dvs. să vă explice cum a gândit un algoritm sau cum a realizat o aplicație! Veți constata supărați că nu înțelegeți mai nimic din cuvintele pe care le ascultați. Copiii foarte repede se adaptează la stilul de a lucra "în liniște" cu calculatorul și se mulțumesc repede cu mesajele scurte pe care acesta i le comunică. Astfel, încet, încet li se atrofiază capacitatea de a se exprima în fraze, în următoarea fază li se pare inutil să folosească cuvintele vocabularului limbii materne și tot mai des recurg la prescurtări sau cuvinte împrumutate din limba engleză sau din alte limbi.

Nu putem lăsa doar pe seama profesorului de limba și literatura română sarcina de a cultiva modul de exprimare a tinerilor în limba lor maternă. Trebuie să creăm cât mai multe ocazii în care copiii să fie nevoiți să se exprime elevat, să vorbească îngrijit, să respecte regulile gramaticale, să fie atenți la topica limbii vorbite. Altfel în anii care vin nu îi vor înțelege nici propriii părinți.

Noua programă școlară conține multe elemente care oferă posibilități variate de ***predare orientată pe proiecte.*** Nu este o invenție proprie, metoda este practicată în multe școli, de mulți profesori. Câștigul mare care se obține în urma aplicării ei se concretizează prin faptul că elevul va înțelege încă din băncile școlii că doar o aplicație finalizată devine un "produs" care poate fi vândut. De asemenea, aplicațiile trebuie să aibă o interfață prietenoasă și o documentație. Și am și ajuns la importanța muncii în echipă, respectiv la modul în care suntem în stare să explicăm ceva.

Clara Ionescu

PROBLEME propuse pentru REZOLVARE

Așa cum veți afla din enunțul problemei "Apărați cetatea", în mijlocul acestei veri, la Mediaș, s-au adunat cei 17 componenți ai Lotului Național și într-o competiție "dură" s-au întrecut pentru a câștiga dreptul de a participa la Balcaniada de Informatică, respectiv la Olimpiada de Informatică a Europei Centrale organizate la Ioannina (Grecia), respectiv la Brno (Cehia).

P069901: A. P. M.

Prof. Dana Lica, Colegiul Național "I.L. Caragiale", Ploiești

Se consideră un arbore având costuri atașate pe muchii și un șir de numere naturale mai mici decât 30000. Să se adauge arborelui un număr maxim de muchii având costuri din numerele date, astfel încât graful obținut să aibă ca arbore parțial de cost minim arborele inițial.

Date de intrare

Pe prima linie a fișierului text **APM.IN** se află numărul natural `n` reprezentând numărul nodurilor din arbore (`n≤100`).

Pe următoarele `n-1` linii se găsesc câte trei numere naturale despărțite printr-un spațiu, sub forma `i j c`, având semnificația: muchia de la `i` la `j` are costul `c`

Pe următoarele linii se află șirul de numere dat, câte un element pe o linie.

Date de ieșire

Pe fiecare linie a fișierului **APM.OUT** se vor scrie trei numere `i j c`, având semnificația: ***muchie adăugată*** între nodurile `i` și `j` de cost `c`. Cele trei numere se vor despărți prin câte un spațiu.

Observații

1. Fiecare element din șir va reprezenta costul unei singure muchii adăugate.
2. Elementele șirului nu sunt neapărat distincte.
3. Între oricare două noduri din graf va exista cel mult o muchie.
4. În cazul în care nu se poate atașa nici o muchie, în fișierul de ieșire se va scrie `0`.
5. În cazul în care problema admite mai multe soluții, în fișier se va scrie una singură.

Exemplu

APM.IN	APM.OUT
4	1 4 5
1 2 2	2 3 3
3 4 4	
3 1 2	
2	
5	
3	

P069902: For ever

Prof. Roxana Tâmplaru, Liceul de Informatică, Craiova

Se consideră următoarea secvență de algoritm, scrisă în pseudocod:

```
pentru v1=1,k execută
  pentru v2=v1,k execută
    pentru v3=v2,k execută
      ...
        pentru vn=vn-1,k execută
          Incrementează;
```

Să se determine numărul de apeluri ale procedurii `Incrementează`.

Date de intrare

Fișierul de intrare **FOREVER.IN** conține două numere naturale `n` (`1≤n≤5000`) și `k` (`1≤k≤5000`) despărțite printr-un singur spațiu, unde:

- `n` reprezintă numărul structurilor **pentru**;
- `k` reprezintă valoarea finală a tuturor structurilor **pentru**.

Date de ieșire

Rezultatul se va scrie în fișierul **FOREVER.OUT**. Acesta este un număr natural și reprezintă numărul de apeluri ale procedurii `Incrementează`.

Exemplu

FOREVER.IN	FOREVER.OUT
2 8	36

P069903: Spre culmi

Cătălin Frâncu, student MIT, Boston, SUA

Se dă un vector cu N≤10000 numere întregi, cuprinse între 1 și 50000. Să se partiționeze acest vector în cât mai puține subșiruri *strict* crescătoare.

Date de intrare

Fișierul **CRESC.IN** conține două linii. Prima linie conține numărul N, iar a doua conține cele N numere, despărțite prin spații.

Date de ieșire

Fișierul **CRESC.OUT** va conține pe prima linie numărul minim K de subșiruri în care se poate partiționa vectorul. Pe fiecare din următoarele K linii se va descrie câte unul din aceste subșiruri, precizând indicii în vector ai elementelor subșirului, în ordine crescătoare, despărțiți prin spații. Ordinea de afișare a subșirurilor este indiferentă.

Exemplu

CRESC.IN	CRESC.OUT
7	2
9 2 4 7 10 11 8	2 3 4 7
	1 5 6

Semnificația ieșirii este: vectorul poate fi partiționat în două subșiruri. Un subșir conține primul, al cincilea și al șaselea element din vector (deci 9-10-11), iar celălalt conține al doilea, al treilea, al patrulea și ultimul element (deci 2-4-7-8).

P069904: Apărați cetatea

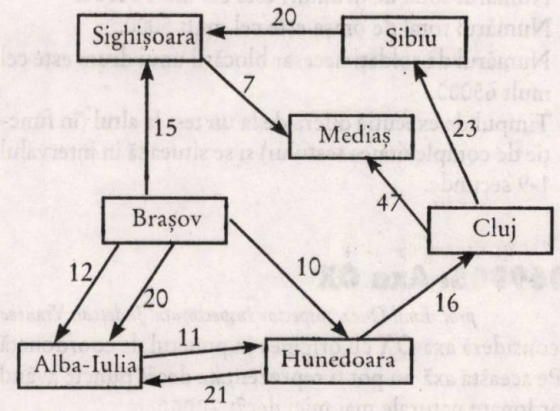
Cristian Cadar, student Universitatea Politehnică, București

Vechiul principat al Transilvaniei este din nou atacat de trupele otomane. De data aceasta, ținta trupelor turcești este Cetatea Mediașului, vestită pentru capetele luminate pe care le adăpostește.

Armata otomană a reușit să intre în Transilvania prin cetatea Brașovului și acum se îndreaptă spre Mediaș. Sarcina voastră, gânditori ai Mediașului, este de a împiedica cucerirea acestui oraș-cetate. Pentru a realiza acest lucru, beneficiați de o hartă a țării, hartă care descrie legăturile dintre orașe. Pentru fiecare drum existent între două orașe, cunoașteți numărul minim de oșteni care trebuie amplasați pe acel drum, pentru a înfrânge turcii care încearcă să treacă pe acolo.

Deoarece armata română este, ca de obicei, redusă, scopul vostru este de a apăra cetatea Mediașului folosind un număr cât mai mic de oșteni.

De exemplu, pentru harta din figura următoare, numărul minim necesar de oșteni este 23, punând 7 oșteni pe drumul de la Sighișoara la Mediaș și 16 oșteni pe drumul de la Hunedoara la Cluj.



Pentru simplificare, vom considera orașele numerotate de la 1 la n, n fiind numărul total de orașe. Orașul 1 va fi considerat Mediașul, iar orașul n Brașovul.

Date de intrare

Pe prima linie a fișierului **MEDIAS.IN** este scris numărul total de orașe, iar pe următoarele linii, până la sfârșitul fișierului sunt scrise câte trei numere i j c, având semnificația: pentru a păzi drumul de la i la j sunt necesari cel puțin c oșteni.

Date de ieșire

Rezultatele se vor scrie în fișierul **MEDIAS.OUT**. Pe prima linie va fi scris numărul minim de soldați necesari, iar următoarele linii, până la sfârșitul fișierului, vor conține câte trei numere naturale i j c, reprezentând faptul că pe drumul de la i la j au fost amplasați c oșteni.

Exemplu

MEDIAS.IN (fișierul corespunde figurii de mai sus)

```
7
3 1 7
6 1 47
7 2 12
7 2 20
2 4 11
7 3 15
5 3 20
7 4 10
4 2 21
4 6 16
6 5 23
```

MEDIAS.OUT

```
23
3 1 7
4 6 16
```

Observaţii

- Drumurile sunt orientate.
- Între două oraşe pot exista mai multe drumuri în ambele direcţii.
- Numărul total de drumuri este cel mult 10000.
- Numărul total de oraşe este cel mult 5000.
- Numărul de soldaţi necesar blocării unui drum este cel mult 65000.
- Timpul de execuţie diferă de la un test la altul (în funcţie de complexitatea testului) şi se situează în intervalul 1-9 secunde.

P069905: Axa OX

prof. Emil Onea, inspector Inspectoratul Judeţean Vrancea

Se consideră axa *OX* cu originea în punctul de coordonată 0. Pe această axă nu pot fi reprezentate decât puncte având coordonate naturale mai mici decât 20000.

Cunoscându-se distanţele dintre *oricare două puncte* reprezentate pe axă se cere determinarea coordonatelor acestora. Lista distanţelor cuprinde cel mult 1000000 de valori.

Date de intrare

În fişierul text **AXA.IN** distanţele dintre punctele reprezentate pe axă sunt scrise pe o linie, despărţite prin câte un spaţiu.

Date de ieşire

În fişierul text **AXA.OUT** se vor scrie coordonatele punctelor reprezentate pe axă, despărţite prin câte un spaţiu.

Observaţii

1. Originea reprezintă un punct de pe axă.
2. Punctele reprezentate pe axă sunt distincte (au coordonate diferite).
3. Problema va admite întotdeauna soluţie.
4. În cazul în care există mai multe soluţii se va afişa una singură.

Exemplu

AXA.IN

```
2 4 6 2 5 7 9 9 11 3
```

AXA.OUT

```
0 2 6 9 11
```

P069906: Oferte promoţionale

Radu Lupşa, asistent Universitatea "Babeş-Bolyai", Cluj

Un lanţ de magazine organizează o acţiune promoţională, oferind gratuit anumite produse clienţilor aflaţi în magazin la anumite ore.

Un client doreşte să obţină un profit cât mai mare de pe urma acestor oferte. În acest scop el pleacă de acasă la momentul 0, îndreptându-se spre primul magazin ales; odată ajuns, aşteaptă ora ofertei, ia gratuit produsul oferit, după care pleacă mai departe spre următorul magazin, şi aşa mai departe. Bineînţeles, drumul de la un magazin la altul îi ia un timp strict pozitiv, iar deplasarea costă, reducându-i din profit. La finalul zilei, adică nu mai târziu de momentul 57600, clientul nostru trebuie să ajungă înapoi acasă.

Se cere o planificare a vizitei magazinelor care să maximizeze profitul.

Date de intrare

Datele de intrare se citesc din fişierul de intrare **PROMOTII.IN**, în care:

- pe prima linie sunt scrise două numere naturale N ($N \leq 100$) şi M ($M \leq 5000$) reprezentând numărul de magazine, respectiv numărul total de oferte (separate printr-un spaţiu);
- pe fiecare din următoarele M linii, sunt scrise câte trei numere: numărul de ordine al magazinului, momentul la care are loc oferta promoţională şi valoarea produsului oferit; nu există două oferte în acelaşi magazin la aceeaşi oră;
- pe următoarele N+1 linii, este scrisă matricea timpilor necesari deplasării. În linia i coloana j se găseşte timpul necesar deplasării de la magazinul i la magazinul j; magazinul cu numărul N+1 reprezentând casa clientului nostru; diagonala matricei conţine zerouri;
- pe următoarele N+1 linii se află matricea costurilor deplasărilor, cu aceleaşi convenţii ca mai sus.

Date de ieşire

Ieşirea se va face în fişierul **PROMOTII.OUT** în următorul format: pe prima linie se va afişa profitul obţinut de client, iar în continuare pentru fiecare obiect promoţional primit se va scrie o linie conţinând două numere: numărul magazinului urmat de momentul la care are loc oferta.

Exemplu

PROMOTII.IN

```
2 5
1 2 10
2 3 20
1 4 25
1 5 10
2 10 10
0 2 2
2 0 3
1 1 0
0 1 1
1 0 1
1 1 0
```

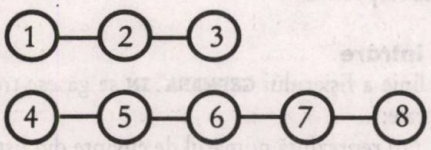
PROMOTII.OUT

```
52
1 2
1 4
1 5
2 10
```

P069907: La grădiniță

Lect. univ Clara Ionescu, Universitatea "Babeș-Bolyai", Cluj

Cu ocazia sfârșitului anului școlar la grădiniță se pregătește un spectacol. Copiii sunt aliniați în două grupuri și așezați față în față. Cei care fac parte din același grup, se țin de mână:



Urmează o acțiune pe care o numim *formarea unui singur lanț de copii* și care se va realiza astfel încât oricare doi copii, aflați în două grupuri diferite, se pot prinde de mână cu condiția să nu își "încrucișeze" mâinile; în figura 1 este vizualizată o posibilitate pentru exemplul dat.

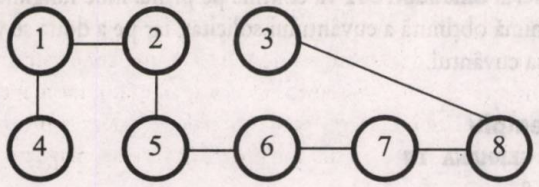


figura 1

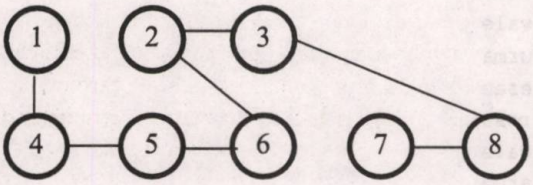


figura 2

Fie `1`, `2`, ..., `m` copiii din primul grup și `m+1`, `m+2`, ..., `m+n` copiii din al doilea grup. Vom da în continuare două semnificații noțiunii de *lanț corect*.

1) Pentru orice `1 ≤ i < j ≤ m`, `i` trebuie să apară în noul lanț înaintea lui `j`. O condiție similară se pune și pentru al doilea grup de copii.
2) Condiția 1) se *relaxează*, în sensul că în momentul în care unul dintre cele două grupuri de copii s-a încheiat, elementele rămase din al doilea grup pot fi considerate și în ordine inversă (figura 2). De asemenea, lanțul poate să înceapă cu copii aparținând inițial unuia din grupuri, așezați în ordine inversă (figura 3). Cele două relaxări se pot combina (figura 4).

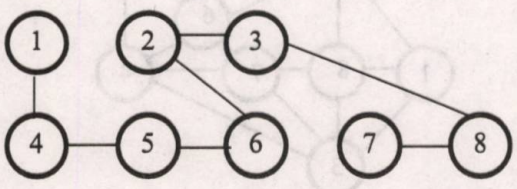


figura 3

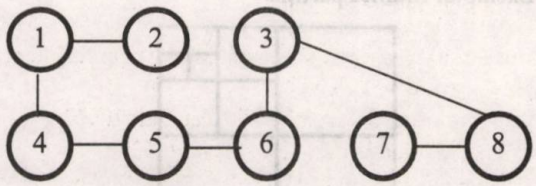


figura 4

Scrieți un program care stabilește numărul lanțurilor corecte care se pot forma respectând regula 1), apoi numărul lanțurilor corecte care se pot forma respectând regula 2).

Date de intrare

Numerele întregi `n` (`1<n≤20`) și `m` (`1<m≤20`), reprezentând numărul copiilor din cele două grupuri se vor citi din fișierul de intrare de tip text **`COPII.IN`**.

Date de ieșire

Numărul lanțurilor corecte care se pot forma respectând regula 1) și numărul lanțurilor corecte care se pot forma respectând regula 2) se vor scrie în fișierul **`COPII.OUT`**.

Exemplu

`COPII.IN`	**`COPII.OUT`**
`2 2`	`6 8`

P069908: Pătrate

prof. Doru PopescuAnastasiu, Liceul "Radu Greceanu", Slatina

Se dă un număr întreg `n` (`6≤n≤100`). Să se determine o modalitate de partiționare a unui pătrat în `n` pătrate.

Date de intrare

Pe prima linie a fișierului **`PATRAT.IN`** se află numărul `n`.

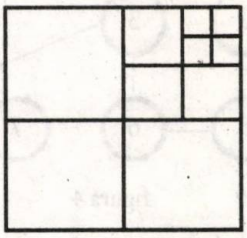
Date de ieșire

În fișierul **`PATRAT.OUT`** se va scrie un tablou pătratic `T` ale cărui elemente sunt numere naturale din mulțimea `M=1,2,...,n`. Fiecare element `x∈M` este folosit în `T` pentru reprezentarea unui singur pătrat din partiție. Liniile tabloului `T` se vor scrie în fișierul de ieșire pe rânduri separate. Pe un rând al fișierului **`PATRAT.OUT`** elementele lui `T` sunt separate printr-un spațiu.

Exemplu

`PATRAT.IN`	**`PATRAT.OUT`**
`10`	`1 1 1 1 4 4 7 8`
	`1 1 1 1 4 4 9 10`
	`1 1 1 1 5 5 6 6`
	`1 1 1 1 5 5 6 6`
	`2 2 2 2 3 3 3 3`
	`2 2 2 2 3 3 3 3`
	`2 2 2 2 3 3 3 3`
	`2 2 2 2 3 3 3 3`

Exemplul codifică partiția:



P069909: Puncte

prof. Doru PopescuAnastasiu, Liceul "Radu Greceanu", Slatina

Se dau n ($3 \le n \le 5000$) puncte în plan astfel încât oricare trei sunt necoliniare. Punctele sunt date prin coordonatele lor carteziene (x_1, y_1); (x_2, y_2); ...; (x_n, y_n), numere întregi din intervalul [-30000, 30000]. Se cere să se determine un cerc (centrul și raza sa) care să treacă prin cel puțin trei din cele n puncte și să nu conțină în interiorul său nici un alt punct din cele n puncte date.

Date de intrare

Fișierul **PUNCTE.IN** are structura:

```
n
x1 y1
x2 y2
...
xn yn
```

Date de ieșire

Fișierul **PUNCTE.OUT** va avea structura:

```
a b r
```

unde a b r (numere reale aproximate la șapte zecimale) sunt coordonatele carteziene ale centrului cercului, respectiv raza.

Exemplu

PUNCTE.IN

```
5
1 8
0 5
3 0
0 7
5 0
```

PUNCTE.OUT

```
4.0000000 4.0000000 4.1231056
```

P069910: La ora de germană

Ovidiu Gheorghioiu, student MIT, Boston, SUA

Se știe că nemții au o adevărată pasiune pentru formarea de noi cuvinte prin contopirea unora existente, uneori rezultând cuvinte foarte lungi, ca de exemplu *Sehenswurdigkeiten* (= lucru demn de văzut, monument).

Profesoara de germană, care pregătește un referat pe această temă, v-a rugat pe voi, lotul de informatică să o ajutați să "prepare" cuvinte noi. Ea v-a dat o listă de cuvinte (care, mai mult sau mai puțin întâmplător, au aceeași lungime), și v-a rugat să îi găsiți un cuvânt nou, cât mai scurt posibil, care să conțină K cuvinte (nu neapărat diferite) din lista respectivă.

Date de intrare

Pe prima linie a fișierului **GERMANA.IN** se găsesc trei numere naturale:

- N ($1 \le N \le 100$) reprezintă numărul de cuvinte din listă,
- L ($1 \le L \le 255$), reprezintă lungimea comună a cuvintelor,
- K ($1 \le K \le 255$), numărul de poziții din cuvântul rezultat pe care trebuie să apară cuvinte din listă. Pe următoarele N linii se găsesc N cuvinte distincte de lungime exact L.

Date de ieșire

Fișierul **GERMANA.OUT** va conține pe prima linie lungimea minimă obținută a cuvântului solicitat, iar pe a doua se va afișa cuvântul.

Exemplu

GERMANA.IN

```
8 4 7
iese
maro
vale
urma
eseu
rosu
oaie
aron
```

GERMANA.OUT

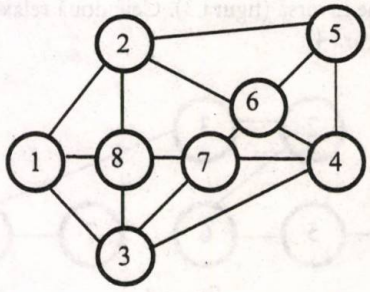
```
16
maroaieseurmaron sau ieseurmaroaieseu
```

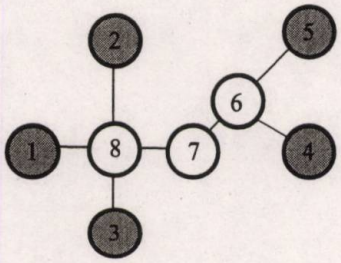
P069911: Arbore parțial

Lect. univ Clara Ionescu, Universitatea "Babeș-Bolyai", Cluj

Se consideră un graf neorientat conex având n noduri. Să se determine unul din arborii parțiali ai grafului dat, care are număr maxim de frunze.

Exemplu





Date de intrare

În fișierul **ARBORE.IN** pe prima linie este scris un număr natural n ($3 \le n \le 2000$) reprezentând numărul nodurilor. Pe următoarele n linii urmează descrierea grafului, care va avea cel mult 10000 de muchii și este conex. Primul număr de pe cea de a j+1-a linie precizează numărul de noduri k, adiacente cu nodul j. După acest număr urmează k numere naturale $x_k \{1,n\} \setminus \{j\}$ reprezentând nodurile adiacente nodului j. Datele scrise pe aceeași linie sunt despărțite prin câte un spațiu.

Date de ieșire

În fișierul **ARBORE.OUT**, pe prima linie se va scrie numărul maxim de frunze. Pe următoarele n-1 linii se vor scrie perechi de numere naturale (despărțite printr-un spațiu) reprezentând muchiile arborelui având acest număr maxim de frunze. În cazul în care există mai multe soluții, se va preciza una singură. Ordinea în care se afișează muchiile și ordinea în care se precizează vârfurile muchiilor este oarecare.

Exemplu

ARBORE.IN

```
8
3 2 3 8
4 1 5 6 8
4 1 4 7 8
4 3 5 6 7
3 2 4 6
4 2 4 5 7
4 3 4 6 8
4 1 2 3 7
```

ARBORE.OUT

```
5
1 8
2 8
3 8
7 8
7 6
4 6
5 6
```

P069912: Numere

prof. Doru PopescuAnastasiu, Liceul "Radu Greceanu", Slatina

Se dau n numere naturale $x_1, x_2, ..., x_n$ din intervalul [1,109]. Să se determine (dacă există), pentru un număr natural m din intervalul [1,109] un șir de numere naturale $a_0, a_1, ..., a_k$ având proprietățile:

1) $k<n$
2) $m=a_0+a_1x_1+a_2x_1x_2+...+a_kx_1x_2...x_k$
3) $a_i<x_{i+1}$, $0 \le i \le k$
4) $a_k \neq 0$

Date de intrare

Pe prima linie a fișierului **NUMERE.IN** este scris numărul m, ($1 \le m \le 109$) reprezentând numărul pentru care se cere descompunerea precizată în enunț. Pe următoarea linie este scris numărul natural n ($1 \le n \le 1000$) reprezentând numărul de numere date. Pe cea de a treia linie se află scrise cele n numere $x_1, x_2, ..., x_n$, despărțite prin câte un spațiu.

Date de ieșire

În cazul în care există soluție, pe prima linie a fișierului **NUMERE.OUT** se va scrie cuvântul DA, în caz contrar se va scrie NU. În caz afirmativ pe a doua linie se va scrie un număr natural k, având semnificația din enunț. Pe cea de a treia linie se vor scrie cele k+1 numere $a_0, a_1, ..., a_k$, despărțite prin câte un spațiu.

Exemple

NUMERE.IN

```
153
5
2 3 5 4 7
```

NUMERE.OUT

```
DA
4
1 1 0 1 1
```

NUMERE.IN

```
150
4
2 3 4 2
```

NUMERE.OUT

```
NU
```

selecție pentru CEOI și
baraje de
SNG
MEDIAȘ
Școala Națională de Gaz
BOI 1999

SOLUȚIILE problemelor propuse

În numărul 5/1999 al GInfo am publicat cele 24 de probleme de la ONI '99, organizată la Mediaș. Redacția GInfo, preocupată de a vă oferi posibilitatea de a "crește" mereu, vă prezintă rezolvările unora dintre ele.

P059901: Expresie

Sabău Florin, clasa a XI-a, Reșița

Procedura `Eval` realizează evaluarea expresiei. Aceasta defalcă expresia stocată în variabila `exp` în părțile componente:
- mai întâi operatorii - în vectorul `o`
- apoi constantele numerice și variabilele - în vectorul `op`

Urmează evaluarea propriu-zisă în doi pași:
- înmulțirea;
- adunarea și scăderea.

Listing EXPRESIE.PAS

```
var exp:string;
    v:array['a'..'e'] of Longint;
    e:array['a'..'e'] of Boolean;
    op:array[1..255] of Comp;
    o:array[1..255] of Char;
    i:Integer;
    s:Comp;

function s2i(s:string):Longint;
var i:Longint;
    code:Integer;
begin
  val(s,i,code);
  s2i:=i
end;

procedure citeste;
var s:string;
    c:Char;
begin
  Assign(input,'EX.IN'); Reset(input);
  Readln(exp);
  Fillchar(e,Sizeof(e),0);
  i:=0;
  while not Eof do
  begin
    Inc(i);
    Readln(s);
    c:=s[1];
    v[c]:=s2i(Copy(s,3,Length(s)-2));
    e[c]:=true
  end;
  Close(input);
  Assign(output,'EX.OUT');
  Rewrite(output)
end;

function nedefinita:Boolean;
begin
  nedefinita:=false;
  for i:=1 to Length(exp) do
    if (exp[i] in ['a'..'e']) and
       (e[exp[i]]=false)
    then
    begin nedefinita:=true; Break end
end;

function eval:string;
var vec,i,n,m,k:Integer;
    sw:Comp;
begin
  i:=1;
  n:=0; m:=0;
  o[1]:='+';
  while i<=Length(exp) do
  begin
    vec:=i;
    while not (exp[i] in ['a'..'e','+','-'])
          and (i<=Length(exp)) do i:=i+1;
    if (vec<>i)
    then
```

```
    begin
      Inc(n);
      op[n]:=s2i(Copy(exp,vec,i-vec));
      if i<=Length(exp)
      then
      if exp[i] in ['a'..'e']
      then o[n+1]:='*'
      else
      begin o[n+1]:=exp[i]; Inc(i) end
    end;
    vec:=i;
    while (exp[i] in ['a'..'e']) and
          i<=Length(exp)) do
    begin
      Inc(n);
      op[n]:=v[exp[i]];
      o[n+1]:='*';
      Inc(i)
    end;
    if (vec<>i)
    then
    begin
      if exp[i] in ['0'..'9']
      then o[n+1]:='*'
      else
      begin o[n+1]:=exp[i]; Inc(i) end
    end
  end;
  o[n+1]:=#0;
  i:=1;
  while i<=n do       { Pasul 1 - Inmultirea }
  begin
    vec:=i;
    while (o[i+1]='*') and (i<=n) do i:=i+1;
    if (i<=n)
    then
    begin
      sw:=1;
      for k:=vec to i do
      begin
        sw:=sw*op[k];
        if k<>vec then o[k]:=#0
      end;
      op[vec]:=sw
    end;
    while (o[i+1]<>'*') and (i<=n) do i:=i+1
  end;
  s:=0;
  for i:=1 to n do
    if o[i]='+'
    then s:=s+op[i]
    else
      if o[i]='-'
      then s:=s-op[i];
    Writeln(s:0:0)
end;
```

```
Begin
  citeste;
  if nedefinita
  then Writeln('NEDEFINITA')
  else eval;
  Close(output)
End.
```

P059902: Nasturi

Sabău Florin, clasa a XI-a, Reșița

Problema nu are soluție pentru următoarele seturi de date:
- numărul de nasturi (x) este impar;
- numărul de nasturi de extras este cu 2 mai mic decât numărul total de nasturi (n*n);
- numărul de nasturi de extras este mai mic decât 4 (=2).

Dacă nici una dintre condiții nu este îndeplinită atunci avem soluție. Dacă numărul x este divizibil cu 4 atunci simpla extragere a unor grupuri de nasturi de câte 4 este suficientă.

```
..
..
```

Dacă nu, se extrag câte 4 nasturi până când mai rămân 6. Aceștia șase sunt extrași din colțul din dreapta-jos al matricei în felul următor:

```
o..
.o.
..o
```

ori, dacă antepenultima linie a matricei este ocupată în întregime, avem:

```
-....
-..oo
-.oo.
-.o.o
```

('.' reprezintă locul de unde se extrag nasturi, iar 'o' nasturi neextrași).

Listing: NASTURI.PAS

```
var n,x,i,j:Integer;
    a:array[1..31,1..31] of Char;
    gata:Boolean;

procedure citeste;
begin
  Assign(input,'NASTURI.IN'); Reset(input);
  Readln(n,x); Close(input);
  Assign(output,'NASTURI.OUT'); Rewrite(output)
end;

Begin
  citeste;
  if (x mod 2<>0) or (x>=n*n-2) or (x<4)
  then Writeln('fara solutie')
  else
  begin
    Fillchar(a,Sizeof(a),'o');
    gata:=false;
```

```
    for i:=1 to n div 2 do
      if not gata
      then
        for j:=1 to n div 2 do
        begin
          if (x=6) or (x=0)
          then
          begin gata:=true; Break end;
          a[2*i-1,2*j-1]:='.';
          a[2*i,2*j-1]:='.';
          a[2*i-1,2*j]:='.';
          a[2*i,2*j]:='.';
          Dec(x,4);
          if (x=6) or (x=0)
          then
          begin gata:=true; Break end
        end;
        if x=6
        then
        begin
          if a[n-3,n]<>'.'
          then
          begin
            a[n-2,n-1]:='.';
            a[n-2,n]:='.';
            a[n-1,n]:='.';
            a[n-1,n-2]:='.';
            a[n,n-2]:='.';
            a[n,n-1]:='.'
          end
          else
          begin
            a[n-2,n-1]:='o';
            a[n-2,n]:='o';
            for i:=n-5 to n-3 do
            begin
              a[n-1,i]:='.';
              a[n,i]:='.';
              a[n-1,n]:='.';
              a[n,n-1]:='.'
            end
          end
        end;
        for i:=1 to n do
        begin
          for j:=1 to n do Write(a[i,j]);
          Writeln
        end
  end;
  Close(output)
End.
```

P059904: Furtună în Balcani

Sabău Florin, clasa a XI-a, Reșița

Pentru această problemă am folosit algoritmul "fill". Pentru început se umple exteriorul "taberelor" cu un alt caracter (`'o'`) pentru a diferenția exteriorul de interiorul taberelor. Se pornește din punctul `(1,1)` deoarece se știe din problemă că taberele nu ating marginea. Urmează apoi o parcurgere a matricei: când se întâlnește sârma ghimpată (`'X'`) se umple exteriorul taberei (caracterul `'o'`) pe direcțiile N, S, V, E cu mine (`'M'`).

În final se tipărește în fișierul de ieșire matricea.

Listing: BALCANI.PAS

```
{$m 65000,0,0}
var m,n,i,j:Integer;
    a:array[1..100,1..100] of Char;
    x,y:Integer;
    cul1,cul2:Char;

procedure citeste;
begin
  Assign(input,'NATO.IN');
  Reset(input);
  Readln(m,n);
  for i:=1 to m do
  begin
    for j:=1 to n do Read(a[i,j]);
    Readln
  end;
  Close(input);
  Assign(output,'NATO.OUT');
  Rewrite(output)
end;

procedure fill;
begin
  if a[x,y]<>cul1 then Exit;
  a[x,y]:=cul2;
  Inc(x); fill; Dec(x);
  Inc(y); fill; Dec(y);
  Dec(x); fill; Inc(x);
  Dec(y); fill; Inc(y)
end;

Begin
  citeste;
  x:=1; y:=1; cul1:='.'; cul2:='o';
  fill;
  for x:=1 to m do
    for y:=1 to n do
    begin
      if a[x,y]='X'
      then
      begin
        if a[x-1,y]='o' then a[x-1,y]:='M';
        if a[x,y-1]='o' then a[x,y-1]:='M';
        if a[x+1,y]='o' then a[x+1,y]:='M';
        if a[x,y+1]='o' then a[x,y+1]:='M'
      end
    end;
```

```
    for x:=1 to m do
    begin
      for y:=1 to n do
        if a[x,y]='o' then Write('.')
                      else Write(a[x,y]);
        Writeln
    end;
    Close(output)
End.
```

P059905: Semne

Sabău Florin, clasa a XI-a, Reșița

Nu se poate obține o expresie în cazul în care n este mai mic decât 3. Pentru n≥3 se observă că folosind succesiunea ++--++--... pentru un număr de forma 4k+3 se obține 0. Pentru un număr de forma 4k+1, folosind aceeași succesiune de operatori, am obține 1, deci trebuie să micșorăm expresia cu o unitate. De aceea vom substitui începutul expresiei +1+2 (=3) cu +1*2 (=2), în rest aceasta rămâne neschimbată. Pentru 4k vom folosi înșiruirea:

+-+-...+--+...-+-+

în care primul și ultimul termen sunt anulați de al doilea și penultimul etc. Pentru 4k+2 vom porni de la aceeași înșiruire de operatori ca și la 4k. Aici avem 2 termeni în plus, a căror diferență este de 1 (sau -1). Dacă pentru acești ultimi termeni se folosește -+, atunci trebuie să anulăm acest 1 la începutul expresiei. Înlocuim deci +1-2 (=-1) cu -1*2 (=-2).

Listing: SEMNE.PAS

```
const semn:array[0..1] of Char=('+','-');
var n:Longint;

procedure citeste;
begin
  Assign(input,'ZERO.IN'); Reset(input);
  Readln(n); Close(input)
end;

function neg(x:Byte):Byte;
begin x:=(x+1) mod 2; neg:=x end;

procedure rezolva;
var i:Longint;
    sem:0..1;
begin
  case (n mod 4) of
  1: begin
       Write('+*'); i:=3; sem:=1;
       while i<n do
       begin
         Write(semn[sem],semn[sem]);
         i:=i+2; sem:=neg(sem)
       end;
       Writeln(semn[sem])
     end;
  2: begin
       Write('-*');
       sem:=0;
       for i:=3 to (n-2) div 2 do
       begin Write(semn[sem]);
             sem:=neg(sem)
       end;
       sem:=neg(sem);
       for i:=1 to (n-2) div 2 do
       begin Write(semn[sem]);
             sem:=neg(sem)
       end;
       Writeln('-+')
     end;
  3: begin
       for i:=1 to n div 4 do Write('++--');
       Writeln('++-')
     end;
  0: begin
       for i:=1 to n div 4 do Write('+-');
       for i:=1 to n div 4 do Write('-+')
     end
  end
end;

Begin
  citeste;
  n:=12;
  if n<3 then Writeln('NU') else rezolva;
  Close(output)
End.
```

P059907: Vile

Andrei Vancea, student Universitatea Tehnică, Cluj

Problema este rezolvată folosind metoda Branch&Bound. Pentru fiecare configurație se reține camera curentă și starea becurilor din cele n camere. Având în vedere faptul că numărul de camere este mai mic decât 16 starea becurilor se poate reține într-un word (un bit pentru fiecare bec). Dintr-o configurație se poate trece în alta prin trecerea în altă cameră sau schimbarea stării unui bec conform condițiilor din enunț. În tabloul ajuns sunt marcate configurațiile care s-au expandat pentru a nu se mai expanda încă o dată.

Existența unui număr mic de configurații (11*2^11) garantează găsirea soluției într-un timp rezonabil.

Listing: VILE.PAS

```
{$M 65384,0,655360}
program vile;
type plista=^lista;
     lista=record
             camera:Byte;
             becuri,nrmut:Word;
             urm,pred:plista;
           end;
```

```
var n,i,c:Byte;
    nr,nr1,nr2:array[1..11] of Byte;
    vec,ap,st:array[1..11,1..11] of Byte;
    ajuns:array[1..11,1..1 shl 11] of Boolean;
    l,k:Word;
    prim,ultim,cr,p:plista;
    gasitsolutie:Boolean;

procedure Citire;
var f:Text;
    i,j:Byte;
begin
  Assign(f,'VILA.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do
  begin
    Read(f,nr[i]);
    for j:=1 to nr[i] do Read(f,vec[i,j])
  end;
  for i:=1 to n do
  begin
    Read(f,nr1[i]);
    for j:=1 to nr1[i] do Read(f,ap[i,j])
  end;
  for i:=1 to n do
  begin
    Read(f,nr2[i]);
    for j:=1 to nr2[i] do Read(f,st[i,j])
  end;
  Close(f)
end;

procedure scriere;
var f:Text;
    i:Byte;
  procedure Mutari(p:plista);
  var i:Byte;
  begin
    if p=prim then Exit;
    Mutari(p^.pred);
    if p^.camera<>p^.pred^.camera
    then Writeln(f,'m',p^.camera)
    else
      for i:=0 to n-1 do
        if (p^.becuri and (1 shl i)=0) and
           (p^.pred^.becuri and (1 shl i)<>0)
        then
        begin Writeln(f,'s',i+1); Break end
        else
          if (p^.becuri and (1 shl i)<>0) and
             (p^.pred^.becuri and (1 shl i)=0)
          then
          begin
            Writeln(f,'a',i+1); Break
          end
  end;
```

```
begin
  Assign(f,'VILA.OUT'); Rewrite(f);
  if cr=nil
  then Writeln(f,'Nu exista solutie')
  else
  begin
    Writeln(f,cr^.nrmut);
    Mutari(cr)
  end;
  Close(f)
end;

Begin
  Citire;
  Fillchar(ajuns,Sizeof(ajuns),0);
  ajuns[1,1]:=true;
  l:=1;
  new(prim);
  prim^.camera:=1; prim^.becuri:=1;
  prim^.nrmut:=0; prim^.urm:=nil;
  prim^.pred:=nil; ultim:=prim;
  cr:=prim; gasitsolutie:=false;
  while cr<>nil do
  begin
    c:=cr^.camera;
    for i:=1 to nr[c] do
      if (cr^.becuri and
          (1 shl (vec[c,i]-1))<>0) and
          (not ajuns[vec[c,i],cr^.becuri])
      then
      begin
        ajuns[vec[c,i],cr^.becuri]:=true;
        New(p);
        p^.urm:=nil;
        p^.pred:=cr;
        p^.camera:=vec[c, i];
        p^.becuri:=cr^.becuri;
        ultim^.urm:=p;
        ultim:=p;
        p^.nrmut:=cr^.nrmut+1;
        if (vec[c,i]=n) and
           (p^.becuri=1 shl (n-1))
        then
          begin
            cr:=p;
            gasitsolutie:=true;
            Break
          end
      end;
    if gasitsolutie then Break;
    for i:=1 to nr1[c] do
    if (cr^.becuri and
        (1 shl (ap[c,i]-1))=0) and
        (not ajuns[c, cr^.becuri+
        (1 shl (ap[c,i]-1))])
```

```
      then
      begin
      ajuns[c, cr^.becuri+
                (1 shl (ap[c,i]-1))]:=true;
    New(p); p^.urm:=nil;
    p^.pred:=cr;
    p^.camera:=cr^.camera;
    p^.becuri:=cr^.becuri+
                      (1 shl (ap[c,i]-1));
    p^.nrmut:=cr^.nrmut+1;
    ultim^.urm:=p; ultim:=p
  end;
  for i:=1 to nr2[c] do
    if (cr^.becuri and
       (1 shl (st[c,i]-1))<>0) and
       (not ajuns[c, cr^.becuri-
       (1 shl (st[c,i]-1))])
    then
    begin
      ajuns[c, cr^.becuri-
                (1 shl (st[c,i]-1))]:=true;
      New(p); p^.urm:=nil; p^.pred:=cr;
      p^.camera:=cr^.camera;
      p^.becuri:=cr^.becuri-
                      (1 shl (st[c,i]-1));
      p^.nrmut:=cr^.nrmut+1;
      ultim^.urm:=p; ultim:=p;
      if (c=n)and(p^.becuri=1 shl (n-1))
      then
      begin
        cr:=p; gasitsolutie:=true;
        Break
      end
    end;
  if gasitsolutie then Break;
  cr:=cr^.urm
 end
End.
```

P059915: Peisaje

Andrei Vancea, student Universitatea Tehnică, Cluj

Problema este rezolvată prin programare dinamică.

Se definesc:

- `a[i,j,1]` - numărul de posibilități ca după trasarea a `i` caractere să se ajungă la înălțimea `j` și, în plus, condiția din enunț să fie îndeplinită (adică există cel puțin un munte cu înălțimea mai mare sau egală cu `k`);
- `a[i,j,0]` - numărul de posibilități ca după trasarea a `i` caractere să se ajungă la înălțimea `j` și condiția din enunț să nu fie îndeplinită.

Soluția problemei va fi `a[2*n,0,1]`. Valorile tabloului `a` se vor calcula în următorul mod:

```
dacă    k=1
atunci  a[1,1,1]=1
altfel  a[1,1,1]=0
a[1,1,0]=1-a[1,1,1];
dacă    j=k
atunci  a[i,j,1]=a[i-1,j-1,0]+a[i-1,j-1,1]+
                                 +a[i-1,j+1,1]
        a[i,j,0]=0
dacă    j<>k
atunci  a[i,j,1]=a[i-1,j-1,1]+a[i-1,j+1,1];
        a[i,j,0]=a[i-1,j-1,0]+a[i-1,j+1,0];
```

Listing: PEISAJE.PAS

```
{$M 16384,0,655360}
program peisaje;
var a:array[-1..62,-1..31,0..1] of Comp;
    f:Text;
    n,k,i,j:Byte;
Begin
  Assign(f,'PEISAJ.IN'); Reset(f);
  Readln(f,n,k);
  Close(f); Fillchar(a,Sizeof(a),0);
  if k=1
  then a[1,1,1]:=1
  else a[1,1,0]:=1;
  for i:=2 to 2*n do
    for j:=0 to n do
    begin
      if j=k
      then a[i,j,1]:=a[i-1,j-1,0]+
                     a[i-1,j-1,1]+a[i-1,j+1,1]
      else
      begin
        a[i,j,1]:=a[i-1,j-1,1]+a[i-1,j+1,1];
        a[i,j,0]:=a[i-1,j-1,0]+a[i-1,j+1,0]
      end
    end;
  Assign(f,'PEISAJ.OUT'); Rewrite(f);
  Writeln(f,a[2*n,0,1]:0:0);
  Close(f)
End.
```

P59923: Un dreptunghi

Andrei Vancea, student Universitatea Tehnică, Cluj

Pentru fiecare linie se contorizează numărul de bile aruncate despre care suntem informați că s-au întors și cele despre care ni se spune că nu s-au întors, apoi calculăm diferența acestora. Această diferență se va reține în șirul `a`. Problema se reduce la determinarea unei subsecvențe de sumă maximă din șirul `a`. Din extremitățile acelei subsecvențe se calculează valorile `L` și `H`. Dacă suma maximală calculată este negativă atunci `H` va fi `0`. Același algoritm se aplică și pentru coloane, calculându-se `C` și `W`.

Listing: DREPTUNGHI.PAS

```
program dreptunghi;
type sir=array[1..20000] of Integer;
var l,c,h,w,m,n,q,r,i,k,b:Word;
    a,s,pred:^sir;
    f:Text;
```

```
Begin
  New(a);
  Fillchar(a^,Sizeof(a^),0);
  New(s);
  Fillchar(s^,Sizeof(s^),0);
  New(pred);
  Fillchar(pred^,Sizeof(pred^),0);
  Assign(f,'DREPT.IN'); Reset(f);
  Readln(f,m,n); Readln(f,q);
  for i:=1 to q do
  begin
    Readln(f,k,b);
    if b=0 then Dec(a^[k])
           else Inc(a^[k])
  end;
  s^[1]:=a^[1];
  pred^[1]:=1;
  for i:=2 to m do
    if s^[i-1]>=0
    then
    begin
      s^[i]:=s^[i-1]+a^[i];
      pred^[i]:=pred^[i-1]
    end
    else
    begin
      s^[i]:=a^[i];
      pred^[i]:=i
    end;
  b:=1;
  for i:=2 to m do
  if (s^[b]<s^[i]) or ((s^[b]=s^[i]) and
     (b-pred^[b]<i-pred^[i]))
  then b:=i;
  l:=pred^[b];
  if s^[b]>=0 then h:=b-pred^[b]+1
              else h:=0;
  Fillchar(a^,Sizeof(a^),0);
  Fillchar(s^,Sizeof(s^),0);
  Fillchar(pred^,Sizeof(pred^),0);
  Readln(f,r);
  for i:=1 to r do
  begin
    Readln(f,k,b);
    if b=0 then Dec(a^[k])
           else Inc(a^[k])
  end;
  s^[1]:=a^[1];
  pred^[1]:=1;
  for i:=2 to m do
    if s^[i-1]>=0
    then
    begin
      s^[i]:=s^[i-1]+a^[i];
      pred^[i]:=pred^[i-1]
    end
    else
    begin s^[i]:=a^[i]; pred^[i]:=i end;
  b:=1;
  for i:=2 to m do
    if (s^[b]<s^[i]) or ((s^[b]=s^[i]) and
       (b-pred^[b]<i-pred^[i]))
    then b:=i;
  c:=pred^[b];
  if s^[b]>=0 then w:=b-pred^[b]+1
              else w:=0;
  Close(f);
  Assign(f,'DREPT.OUT'); Rewrite(f);
  Writeln(f,l,' ', c,' ',h,' ',w);
  Close(f)
End.
```

P059924: Submulțimi

Andrei Vancea, student Universitatea Tehnică, Cluj

Se traversează șirul și se marchează în tabloul a sumele ce se pot obține din submulțimi ale șirului cu elemente având indicele mai mic sau egal decât indicele elementului curent. Pentru fiecare sumă astfel marcată se reține elementul cu indice maxim al unei submulțimi având acea sumă. Dacă suma ce trebuie marcată a fost marcată înainte de a se ajunge la elementul curent înseamnă că s-a găsit o soluție și se afișează. Deoarece tabloul a are dimensiunea de 400000 de octeți (nu încape într-un segment de memorie) se reprezintă ca un șir de pointeri. Se observă că algoritmul funcționează mai rapid dacă șirul este ordonat crescător. De aceea se ordonează înainte.

Listing: SUBM.PAS

```
program submultimi;
type sir = array[0..49999] of Byte;
var a:array[1..30] of Longint;
    s:array[0..7] of ^sir;
    l1,l2,m,i:Byte;
    j,max,maxx,k,suma:Longint;
    sol1,sol2:array[1..30] of Byte;

procedure citire;
var f:Text;
    i:Byte;
begin
  Assign(f,'SUB.IN'); Reset(f);
  Readln(f,m);
  suma:=0;
  for i:=1 to m do
  begin
    Read(f,a[i]);
    Inc(suma,a[i])
  end;
  Close(f)
end;
```

```
procedure scriere;
var f:Text;
    i:Byte;
begin
  Assign(f,'SUB.OUT'); Rewrite(f);
  Writeln(f,l1);
  for i:=1 to m do
    if sol1[i]=1 then Write(f,a[i], ' ');
  Writeln(f);
  Writeln(f,l2);
  for i:=1 to m do
    if sol2[i]=1 then Write(f,a[i],' ');
  Writeln(f);
  Close(f);
  Halt
end;

procedure scrienu;
var f:Text;
begin
  Assign(f,'SUB.OUT'); Rewrite(f);
  Writeln(f,'NU');
  Close(f);
  Halt
end;

Begin
  Citire;
  for i:=0 to 7 do
  begin
    New(s[i]);
    Fillchar(s[i]^,Sizeof(s[i]^),0)
  end;
  s[0]^[0]:=31;
  max:=0;
  for i:=1 to m-1 do
    for j:=i+1 to m do
      if a[i]>a[j]
      then
      begin
        a[i]:=a[i] xor a[j];
        a[j]:=a[i] xor a[j];
        a[i]:=a[i] xor a[j]
      end;
  for i:=1 to m do
  begin
    Writeln(i);
    maxx:=max;
    for j:=0 to max do
     if (s[j div 50000]^[j mod 50000]<>0) and
        (s[j div 50000]^[j mod 50000]<>i)
      then
      begin
        if s[(j+a[i]) div 50000]^[(j+a[i])
                                  mod 50000]=0
        then
        begin
          s[(j+a[i]) div 50000]^[(j+a[i])
                                  mod 50000]:=i;
          if (j+a[i]>maxx)
          then maxx:=j+a[i];
        end
        else
        begin
          Fillchar(sol1,Sizeof(sol1),0);
          Fillchar(sol2,Sizeof(sol2),0);
          l1:=0;
          l2:=0;
          k:=j;
          sol1[i]:= 1;
          while k<>0 do
          begin
            sol1[s[k div 50000]^
                              [k mod 50000]]:=1;
            Dec(k,a[s[k div 50000]^
                                [k mod 50000]])
          end;
          k:=j+a[i];
          while k<>0 do
          begin
            sol2[s[k div 50000]^
                              [k mod 50000]]:=1;
            Dec(k,a[s[k div 50000]^
                                [k mod 50000]])
          end;
          for k:=1 to m do
            if (sol1[k]=1) and (sol2[k]=1)
            then
            begin
              sol1[k]:=0;
              sol2[k]:=0
            end
            else
              if sol1[k]=1
              then Inc(l1)
              else
                if sol2[k]=1
                then Inc(l2);
            scriere
          end
        end;
    max:=maxx;
    if max>suma div 2
    then max:=suma div 2;
  end;
  scrienu
End.
```

Algoritmi de "VACANȚĂ"

Sorin Groza

Bucuria cea mare a elevilor și profesorilor de pretutindeni, de când există școală pe Pământ, este vacanța... Dar, poate "școlerul" sau "profu' de info" să uite de ceea ce îl pasionează mai mult, și anume de algoritmi? Evident nu!

Pedeapsa

Pentru că a obținut rezultate slabe la matematică în anul școlar încheiat, Prichindel a primit pe perioada vacanței următoarea pedeapsă: va trebui să scrie pe caiet toate numerele din intervalul `[1,n]`, unde `n` este precizat de profesor. După ce a făcut acest lucru va trebui să spună profesorului câte cifre a scris în total și câte dintre numere au fost numere prime. Ajutați-l pe Prichindel scriind programul care rezolvă această problemă, mai ales că s-a zvonit că oferă înghețată celui care îl ajută.

Exemplu

Date de intrare

```
n=100
```

Date de ieșire

```
Prichindel a scris in total 192 cifre
Au fost intalnite 25 numere prime
```

Pentru a rezolva această problemă, pentru fiecare număr de la 2 la `n` va trebui să-i determinăm numărul de cifre și să verificăm dacă este prim. În ceea ce privește problema determinării numărului de cifre, vom folosi unele funcții și proceduri specifice limbajului Pascal, `Str` pentru a converti numărul în șir de caractere și apoi `Length` pentru a afla numărul de caractere adică numărul de cifre a numărului. Pentru a verifica dacă numărul este prim sau nu, vom împărți numărul (dacă acesta este impar) la toate numerele începând cu 3 și până la radicalul numărului analizat. Dacă un număr are divizori, cel puțin unul dintre ei se întâlnește până la radicalul numărului respectiv. Dacă nu se găsește nici un divizor, înseamnă că numărul este prim, situație în care vom incrementa un contor. Atenție!!! numărul 2 este singurul număr par care este prim și pentru numărul 1 nu se pune problema primalității.

Listing: PEDEAPSA.PAS

```
var nr_prime,n,i,nr:Integer;
    total_cifre:Longint;
    este:Boolean;
    sir:string;
Begin
  Write('Ce numar a primit Prichindel?');
  Readln(n);
  total_cifre:=1;
  for nr:=2 to n do
  begin
    Str(nr,sir);
    total_cifre:=total_cifre+Length(sir);
    i:=3;
    este:=Odd(nr) or (n=2);
    while (i<=Sqrt(nr)) and este do
    begin
      if nr mod i=0 then este:=false; Inc(i,2)
    end;
    if este then Inc(nr_prime)
  end;
  Write('Prichindel a scris in total ');
  Writeln(',total_cifre,' cifre');
  Write('Au fost intalnite ');
  Writeln(nr_prime,' numere prime')
End.
```

Excursia

La sfârșit de săptămână o clasă de elevi, împreună cu profesorul lor de informatică, au plecat într-o excursie la munte. După ce au sosit la cabană și s-a instalat fiecare, profesorul a anunțat adunarea în fața cabanei și alinierea pe un rând, în vederea stabilirii sarcinilor. Profesorul (cu gândul mereu la informatică) observă că elevii s-au așezat într-un mod foarte interesant: așezarea lor are aspect de "munte", adică toți copiii până la cel de pe o poziție `k` (numit vârful muntelui) au înălțimile în ordine strict crescătoare, iar în continuare, până la sfârșit, în ordine strict descrescătoare. Profesorul cere celui mai bun informatician din clasă să scrie un program care verifică dacă așezarea copiilor are sau nu aspect de "munte". În caz afirmativ programul va afișa mesajul `"Asezare cu aspect de munte cu varful la"` urmat de poziția copilului ce reprezintă "vârful muntelui", în caz contrar se va afișa mesajul `"Asezare oarecare"`.

Exemplu
Date de intrare
6 - numărul copiilor
9 14 36 71 10 4 - înălțimile copiilor
Date de ieșire
Asezare cu aspect de munte cu varful la 4

Date de intrare
7
15 47 21 13 26 82 45
Date de ieșire
Asezare oarecare

Este clar că dacă așezarea are aspect de munte, vârful este elevul cu înălțime maximă din șir. Vom afla maximul și poziția lui în șirul de elevi. Dacă poziția maximului este prima sau ultima, atunci așezarea nu are aspect de munte. Se verifică în continuare cele două părți, prima de la început și până la poziția maximului și a doua de la poziția maximului și până la sfârșit. Dacă prima parte nu este crescătoare sau cea de-a doua nu este descrescătoare, atunci așezarea este oarecare. În caz contrar așezarea are aspect de munte cu vârful la poziția maximului.

Listing: ASEZARE.PAS
```
var i,a,j,max,n,sir:Integer;
    x:array[1..100] of Integer;
    bun:Boolean;
begin
  Write('introduceti numarul de elevi ');
  Readln(n);
  Writeln('Dati inaltimile elevilor ');
  for i:=1 to n do read(x[i]);
  max:=x[1]; a:=1;
  for i:=1 to n do
   if x[i]>max then begin max:=x[i]; a:=i end;
  if (a=1) or (a=n)
  then Writeln('Asezare oarecare')
  else
  begin
    i:=1; bun:=true;
    while (i<=a-1) and (bun) do
    begin
      if x[i]>x[i+1] then bun:=false; Inc(i)
    end;
    j:=a;
    while (j<=n-1) and (bun) do
    begin
      if x[j]<x[j+1] then bun:=false; Inc(j)
    end;
    if bun
     then Writeln('Asezare cu aspect de munte'
                  +'cu varful la ',a)
    else Writeln('Asezare oarecare')
  end
End.
```

Alegerea cabinelor

Iată-ne plecați și în excursie la Geoagiu Băi la o scaldă binemeritată pe această căldură sufocantă. Pe drum, viitorii noștri informaticieni și-au pus întrebarea dacă își pot alege cabine cu numere consecutive dar care să nu fie numere prime. Ei știu că sunt n elevi plecați în excursie și că la Geoagiu sunt m cabine. Scrieți un program care, în caz afirmativ, va afișa toate posibilitățile de alegere a cabinelor conform restricțiilor. O posibilitate va fi indicată prin numerele de ordine ale primei și ultimei cabine. Dacă nu se poate găsi o astfel de posibilitate, se va afișa mesajul Alegere imposibila.

Exemplu
Date de intrare
n = 5
m = 50
Date de ieșire
24 - 28
32 - 36

Deoarece aveau timp suficient la dispozitie, elevii au luat "la rost" fiecare secvență de n numere naturale consecutive. S-a verificat fiecare număr dacă este prim și, în cazul în care s-a întâlnit unul, secvența a fost abandonată. În cazul în care toate numerele din secvență nu sunt prime s-a găsit o posibilitate. Aceasta se va afișa, iar variabila solutie va deveni true pentru a semnala că există soluție. Evident că această soluție nu este una optimă, deoarece se repetă unele calcule. Vă propunem să căutați și alți algoritmi, mai eficienți (eventual folosind șirul sumelor parțiale, discutat în numerele anterioare).

Listing: ALEGERE.PAS
```
var n,m,k,i,j,d,p:Integer;
    este,solutie:Boolean;
Begin
  Write('Cati elevi sunt in excursie... ');
  Readln(n);
  Write('Cate cabine sunt la Geoagiu... ');
  Readln(m);
  for i:=2 to m-n+1 do
  begin
    p:=0;
    j:=i;
    while (j<=i+n-1) and (p=0) do
    begin
      d:=2; este:=true;
      while (d<=sqrt(j)) and (este) do
      begin
        if j mod d=0 then este:=false;
        Inc(d)
      end;
      if este then Inc(p);
      Inc(j)
    end;
```

```
    if p=0 then
    begin
      Writeln(i,' - ',i+n-1);
      solutie:=true
    end
  end;
  if not solutie
  then Writeln('Alegere imposibila')
End.
```

La taifas

Era odată o țară fără griji în care oamenii nu făceau nimic și discutau toată ziua. Ei se așezau într-un șir ce avea la început înțeleptul urmat în continuare de ceilalți, în ordinea descrescătoare a inteligenței lor. Autoritățile, vrând să evite orice situație neplăcută, au hotărât să-i separe, punând oameni de ordine după fiecare grup format din p persoane. Plasarea oamenilor de ordine începe de la cel mai din dreapta și fiecare nou grup de oameni de ordine amplasat are cu unul mai multe persoane decât celălalt, primul fiind format dintr-o singură persoană. Scrieți un program care citește șirul care conține inițialele persoanelor și valoarea p, apoi afișează:

- configurația șirului după amplasarea oamenilor de ordine;
- un mesaj din care să rezulte dacă înțeleptul va fi izolat sau nu, adică va mai avea sau nu un vecin cu care să discute.

Exemplu

Date de intrare

```
abcdbdahb, p=2                    abcde, p=3
```

Date de ieșire

```
a****bc***db**da*hb               ab*cde
izolat                            neizolat
```

Observație: Prin * s-a simbolizat un om de ordine.

În tabără

În tabăra *Pescăruș* (foarte solicitată) au sosit mulți copii. Directorul taberei are o problemă pentru că nu știe câți copii au sosit în tabără. Dar el observă următorul lucru: dacă ar face grupuri de câte n copii ar rămâne p copii (p<n), iar dacă ar face grupuri de câte m copii ar rămâne q copii (q<m). Care este numărul copiilor sosiți în tabără, știind că el este cel mai mic număr care verifică aceste proprietăți. În caz că acest număr nu există se va afișa mesajul `Eroare!!!`

Exemplu

Date de intrare

```
n=5,p=2
m=7,q=4
```

Date de ieșire

```
numarul de copii = 32
```

Date de intrare

```
n=3, p=1
m=3, q=2
```

Date de ieșire

```
Eroare!!!
```

Competiția

Se consideră două clase de elevi, fiecare având n elevi. Elevii din fiecare clasă sunt numerotați cu numere de la 1 la n, neexistând doi elevi cu același număr. Elevii fiecărei clase se așează la întâmplare în șir. Pentru fiecare clasă se va calcula un punctaj astfel: pentru fiecare elev se numără câți de la început până la el au numere mai mari decât el. Se va face suma acestor valori calculate pentru fiecare elev. Suma finală va fi punctajul clasei. Să se calculeze punctajul fiecărei clase și să se afișeze clasa cu punctaj mai mare.

Exemplu

Date de intrare

```
n=6
2,3,1,6,5,4   - clasa I
4,6,1,3,2,5   - clasa II
```

Date de ieșire

```
Punctaj clasa I: 5
Punctaj Clasa II: 8
Castiga clasa II
```

Să jucăm rummy

Un copil are pe o tablă de rummy mai multe piese ce alcătuiesc un număr. Prin scoaterea unei piese se formează două numere noi. În timp ce aștepta mutarea partenerului, și-a pus următoarea problemă: a câta piesă trebuie eliminată astfel încât suma celor două numere obținute să fie maximă. Scrieți un program care, plecând de la numărul de pe tablă, afișează a câta piesă se elimină; de asemenea este afișată suma maximă obținută în această situație.

Exemplu

Date de intrare

```
n = 2735
```

Date de ieșire

```
a 2-a piesă - suma 37
```

Domnul Sorin Groza este profesor la Clubul Copiilor din Aiud și poate fi contactat prin intermediul redacției.

Memorii CACHE

Lucian Vinţan

Acest articol încearcă să prezinte într-o manieră sintetică unul dintre cele mai fecunde concepte ale tehnologiei computerelor, anume acela de memorie cache. Cache-urile au un rol esenţial, anume acela de "acoperire a gap-urilor tehnologice" între diversele componente ale sistemului ierarhizat de memorii.

Cache: a safe place for hiding or storing things.
(Webster's New World Dictionary of the American Language, Third College Edition - 1988)

Memoria cache este o memorie situată din punct de vedere logic între *CPU* (*Central Processing Unit* - unitate centrală) şi memoria principală (uzual *DRAM - Dynamic Random Access Memory*), mai mică, mai rapidă şi mai scumpă (per byte) decât aceasta şi gestionată - în general prin hardware - astfel încât să existe o cât mai mare probabilitate statistică de găsire a datei accesate de către CPU, în cache. Aşadar, cache-ul este adresat de către CPU în paralel cu memoria principală (MP): dacă data dorită a fi accesată se găseşte în cache, accesul la MP se abortează, dacă nu, se accesează MP cu penalizările de timp impuse de latenţa mai mare a acesteia, relativ ridicată în comparaţie cu frecvenţa de tact a CPU. Oricum, data accesată din MP se va introduce şi în cache.

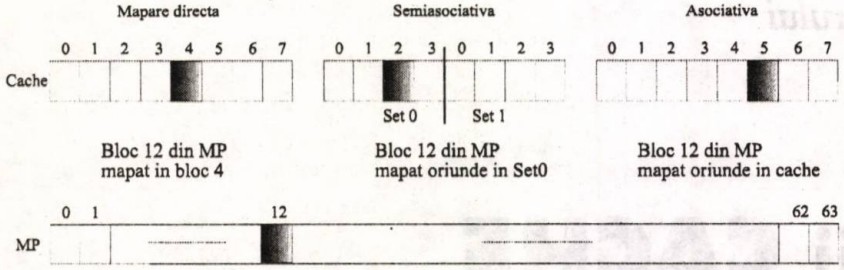
Memoriile cache sunt implementate în tehnologii de înaltă performanţă, având deci un timp de acces foarte redus (cca. 2-7ns la ora actuală). În prezent presiunea asupra acestor memorii este foarte ridicată, rolul lor fiind acela de a apropia performanţa microprocesoarelor (care creşte cu cca. 50-60% pe an) cu aceea a memoriilor DRAM, a căror latenţă scade cu doar cca. 7% pe an. În general, pentru a accesa o locaţie DRAM, un procesor "pierde" 10-30 de impulsuri de tact (~ timp acces DRAM/T_{CLK}, T_{CLK}= *perioada ceasului microprocesorului*), în schimb accesarea cache-ului se face în doar 1-3 impulsuri de tact. Cu alte cuvinte, memoria cache reduce timpul mediu de acces al CPU la MP, ceea ce este foarte util.

Se defineşte un acces al CPU cu *hit* în cache, un acces care găseşte o copie în cache a datei accesate. Un acces cu *miss* în cache este unul care nu găseşte o copie în cache a datei accesate de către CPU şi care, prin urmare, adresează MP cu toate penalizările de timp care derivă din accesarea acesteia.

Se defineşte ca parametru de performanţă al unei memorii cache *rata de hit*, ca fiind raportul statistic între numărul acceselor cu hit în cache şi numărul total al acceselor CPU la memorie. Măsurat pe benchmark-uri (programe de test) reprezentative, la ora actuală sunt frecvente rate de hit de peste 90%. *Rata de miss* (RM) este complementara ratei de hit (RH), astfel că: RH [%] + RM [%]=100 %. În esenţă, utilitatea cache-ului derivă din următorul fapt: la o citire cu miss (din MP), data adusă din MP este introdusă şi în cache, în speranţa că la o următoare citire a aceleiaşi date, aceasta se va găsi în cache (hit). În realitate, în cazul unei citiri cu miss în cache se aduce din MP nu doar data (cuvântul) dorită de către CPU ci un întreg bloc (4-16 cuvinte) care evident conţine data respectivă. O citire cu miss presupune aducerea blocului din MP dar înainte de aceasta se impune evacuarea în MP a unui bloc din cache. Aşadar, transferul din cache în MP se face tot la nivel de bloc şi nu de cuvânt. Astfel, se optimizează traficul între cache şi MP pe baza a două principii care vor fi discutate în continuare.

În esenţă, eficienţa memoriilor cache se bazează pe două principii de natură statistică şi care caracterizează intrinsec noţiunea de program: principiile de localitate temporală şi spaţială. Conform principiului de localitate temporală, există o mare probabilitate ca o dată (instrucţiune) accesată acum de către CPU să fie accesată din nou, în viitorul imediat. Conform principiului de localitate spaţială, există o mare probabilitate ca o dată situată în imediata vecinătate a unei date accesate curent de către CPU, să fie şi ea accesată în viitorul apropiat (pe baza acestui principiu statistic se aduce din MP în cache un întreg bloc şi nu doar strict cuvântul dorit de către CPU). O buclă de program (structură esenţială în orice program) exemplifică clar aceste principii şi justifică eficienţa conceptului de cache.

O combinare a celor două principii anterior expuse conduce la celebra "regulă 90/10" care spune că cca. 90% din timpul de rulare al unui program se execută doar cca. 10% din codul acestuia. Personal, cred că mai puţin. Pe



Este evident că într-un astfel de cache rata de interferență se reduce odată cu creșterea gradului de asociativitate *N*. Aici, de exemplu, blocurile 12, 20, 28 și 36 pot coexista în setul 0. Prin reducerea posibilelor interferențe ale blocurilor, creșterea gradului de asociativitate determină îmbunătățirea ratei de hit și deci a performanței globale. Pe de altă parte însă, asociativitatea impune căutarea după conținut (se caută deci într-un set dacă există memorat blocul respectiv) ceea ce conduce la complicații structurale și deci la creșterea timpului de acces la cache și implicit la diminuarea performanței globale. Optimizarea gradului de asociativitate, a capacității cache, a lungimii blocului din cache etc., nu se poate face decât prin laborioase simulări software, variind toți acești parametri în vederea minimizării ratei globale de procesare a instrucțiunilor (instrucțiuni/ciclu).

baza acestor principii empirice se situează întreg eșafodajul conceptului de cache; eficiența sa deosebită nu poate fi explicată prin considerente analitice pentru simplul fapt că este practic imposibil a descrie analitic noțiunea de program. În fond, ce este un program? Care este distribuția instrucțiunilor sau a primitivelor structurale într-un program? Poate fi aceasta descrisă concret, pe baza unor modele deterministe sau aleatoare? Dificultatea unor răspunsuri exacte la aceste întrebări - dată în fond de imposibilitatea punerii în ecuație a minții umane, cea care creează infinita diversitate de "programe" - face ca cea mai bună explicație asupra eficienței memoriilor cache să stea în cele două principii empirice anterior schițate, caracterizând intrinsec noțiunea de program.

Din punct de vedere arhitectural, există trei tipuri distincte de memorii cache în conformitate cu gradul de asociativitate: *cu mapare directă, semiasociative și total asociative.*

La cache-urile cu mapare directă, ideea principală constă în faptul că un bloc din MP poate fi găsit în cache (hit) într-un bloc unic determinat. În acest caz regula de mapare a unui bloc din MP în cache este:

(Adresa bloc MP) modulo (Nr. blocuri din cache)

Strictețea regulii de mapare conduce la o simplitate constructivă a acestor memorii, dar și la fenomenul de interferență al blocurilor din MP în cache. Astfel, de exemplu, blocurile 12, 20, 28, 36, 42 etc. nu pot coexista în cache la un moment dat întrucât toate se mapează pe blocul 4 din cache. Prin urmare, o buclă de program care ar accesa alternativ blocurile 20 și 28 din MP ar genera o rată de hit egală cu zero.

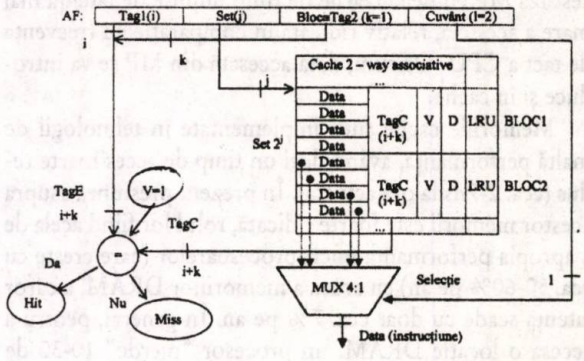
La cache-urile semiasociative există mai multe seturi, fiecare set având mai multe blocuri componente. Aici, regula de mapare precizează strict doar setul în care se poate afla blocul dorit, astfel:

(Adresa bloc MP) modulo (Nr. seturi din cache)

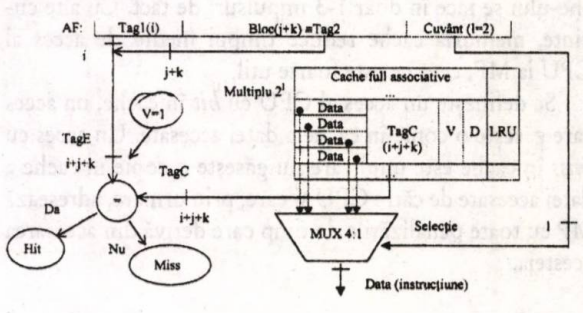
În principiu, blocul dorit se poate mapa oriunde în setul respectiv. Mai precis, la un miss în cache, înainte de încărcarea noului bloc din MP, trebuie evacuat un anumit bloc din setul respectiv. În principiu există implementate două tipuri de algoritmi de evacuare: pseudorandom (cvasialeator) și *LRU* (*Least Recently Used*). Algoritmul *LRU* evacuează blocul din cache cel mai demult neaccesat, în baza principiului de localitate temporală (aflat oarecum în contradicție cu o probabilistică markoviană care ar sugera să fie păstrat!). Dacă un set din cache-ul semiasociativ conține *N* blocuri atunci cache-ul se mai numește "*tip N-way set associative*".

Următoarele 3 scheme prezintă implementări realizate pentru tipurile de cache anterior discutate.

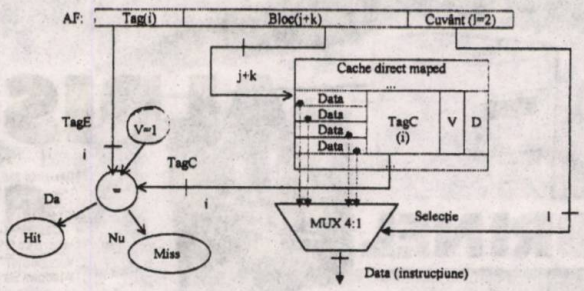
Cache 2-way set associative



Cache full associative



Cache direct mapat



S-a considerat un bloc compus din patru cuvinte. Bitul *V* este un bit de validare a blocului, *V*=1 fiind o condiție necesară a obținerii hitului. Bitul este util îndeosebi în sistemele multiprocesor în vederea menținerii coerenței memoriilor cache locale datorită redundanței informaționale. Mai precis, aici apare necesitatea citirii din cache-ul propriu a ultimei copii modificate a datei respective. Când un procesor modifică o copie locală a unei date, toate blocurile care conțin acea dată din cadrul celorlalte procesoare, trebuie invalidate prin resetarea *V*=0. Necesitatea invalidării blocurilor (*V*=0) apare chiar și în sistemele uniprocesor. Imediat după resetarea sistemului, uzual, procesorul execută un program încărcător rezident în memoria EPROM. Cum imediat după inițializarea sistemului conținutul cache-ului e practic aleator, pentru a evita false hituri la citirea programului încărcător din EPROM, se inițializează biții *V* cu zero. La prima încărcare a unei date (instrucțiuni) în cache, bitul *V* aferent se va seta pe '1', validând astfel hitul.

Bitul *D* (*Dirty*) este pus pe '0' la încărcarea inițială a blocului în cache. La prima scriere a acelui bloc, bitul se pune deci pe '1'. Evacuarea propriu-zisă a blocului se face doar dacă bitul *D* are valoarea 1. Prin acest bit se minimizează evacuările de blocuri în MP, pe baza principiului că un bloc trebuie evacuat numai dacă a fost scris în cache.

În acest sens, din punct de vedere al acceselor de scriere a unui procesor, există două posibilități:

a) Strategia "*Write Through*" (*WT*), prin care informația este scrisă de către procesor atât în blocul aferent din cache cât și în blocul corespunzător din memoria principală.
b) Strategia "*Write - Back*" (*WB*), prin care informația este scrisă numai în cache, blocul modificat fiind transferat în MP numai la evacuarea din cache.

În vederea menținerii coerenței cache-urilor cu precădere în sistemele multimicroprocesor - există 2 posibilități în funcție de ce se întâmplă la o scriere:

1. *Write Invalidate* - prin care CPU care scrie determină ca toate copiile din celelalte memorii cache să fie invalidate înainte ca el să-și modifice blocul din cache-ul propriu.
2. *Write Broadcast* - CPU care scrie pune data de scris pe busul comun spre a fi actualizate toate copiile din celelalte cache-uri.

Ambele strategii de menținere a coerenței pot fi asociate cu oricare dintre protocoalele de scriere (WT, WB) dar de cele mai multe ori se preferă WB cu invalidare. Nu detaliem aici problemele de coerență întrucât acestea se referă cu deosebire la problematica sistemelor multiprocesor și deci depășesc cadrul acestei prezentări.

Apar posibile patru procese distincte într-un cache ca în tabelul următor:

Tip acces	*Hit/Miss*	*Acțiune în cache*
Citire	miss	Evacuare bloc + încărcare bloc nou
Citire	hit	(comparare tag-uri)
Scriere	miss	Evacuare bloc + încărcare bloc nou + scriere dată în bloc
Scriere	hit	Scriere dată în blocul din cache (WB)

Așadar, memoriile cache îmbunătățesc performanța îndeosebi pe citirile cu hit, iar în cazul utilizării scrierii tip *Write Back* și pe scrierile cu hit.

Îmbunătățirea accesului la memorie pe citirile CPU este normală având în vedere că acestea sunt mult mai frecvente decât scrierile (orice instrucțiune implică cel puțin o citire din memorie pentru aducerea sa; statistic, cca. 75 % din accesele la memorie sunt citiri).

Explicația la cauzele miss-urilor în cache-uri, conform literaturii acestui domeniu, sunt de trei tipuri:

- datorită faptului că în fond primul acces la un bloc generează întotdeauna miss (compulsory); sunt inevitabile.
- datorită capacității limitate a cache-ului care nu poate conține la un moment dat toate blocurile din MP, ceea ce implică evacuări/încărcări (capacity).
- datorită interferențelor (conflictelor) unor blocuri din MP pe același bloc din cache (conflict); acestea se reduc odată cu creșterea capacității și a gradului de asociativitate.

Primul care a pus în lumină conceptul de memorie cache a fost prof. *Maurice Wilkes* (Univ. Cambridge, Anglia) - un pionier al calculatoarelor care a inventat în 1951 și tehnica microprogramării unităților de comandă aferente procesoarelor - într-un articol publicat în 1965 ("Slave memories and dynamic storage allocation", IEEE Trans. Electronic Computers, April, 1965). Prima implementare a unui cache (cu mapare directă) aparține probabil lui *Scarrott*, în cadrul unui sistem experimental construit tot la Universitatea din Cambridge. Primul sistem comercial care utiliza cache-urile a fost IBM 360/85 (1968). Conceptul de cache s-a dovedit a fi foarte fecund nu numai în hardware dar și în software, prin aplicații diverse în sistemele de operare (memoria virtuală), rețele de calculatoare, baze de date, compilatoare etc.

Conf.dr.ing. Lucian Vințan este cadru didactic la Facultatea de Inginerie din Sibiu. Poate fi contactat prin e-mail la adresa vintan@cs.sibiu.ro

NORTON CleanSweep
NORTON Utilities
NORTON AntiVirus
EasyRestore
PartitionMagic
LOST & FOUND
ROMSYM DATA-SRL
SOFTWARE
ALPIS
BRINEL
ALPIS®
Designed for Microsoft Windows NT Windows 98
WINDOWS 2000 READY PCs
ISO-9002
COMPAQ
P/N : 131015-a82
Deskpro EN SFF C466
6.4 GB Hard
64 MB RAM
NIC / Sound / W95
Monitor 15"
MAINBOARD
QDI WinneX 1, Intel 82810,
Intel Celeron 400PPGA W/128K cache
32 MB SDRAM DIMM PC 100 (8ns)
SVGA OB BOARD 4MB AGP
HDD 6.4 Quantum Fireball
CD-ROM 40x IDE
SOUND CARD on board +
Stereo Speakers 60W PMPO
FLOPPY 3.5", 1.44MB
ACER PS2 Keyboard WIN'95 +
Mouse SERIAL 3b + Pad
MONITOR COLOR
SVGA SMILE 14" DIGITAL
CORELDRAW 8
OMNILOGIC BGS
SALIENT
SALIENT Romania
MULTIMEDIA KIT
• CD-ROM CREATIVE LABS
• SOUND CARD 16bit
• BOXE 80W
SPRINT
MONITOR SPRINT 17" • model MP 1770N
PixelView
PixelView PlayTVpro
PV-BT878P+(W/FM)
Watch TV +Play TV+ • Motion Capture+
Teletext+ • Closed Capture+
FM Radio + • Remote Control+
Video Camera+ • Video Conferencing Ready
oferit de
AGER
THE COMPUTER COMPANY
Anul 2000 se lasă cu
PREMII
COMPUTER PRESS AGORA

Microsoft® România

Microsoft Office 2000 Professional

TORNADO SISTEME

Canon

Canon Bubble Jet
Color Printer BJC-4400

Iomega Zip Drive
250 MB

GT Grup Transilvae

Procesor: AMD-K6 400 MHz 3DNow!
64 MB RAM • HD 6,7 Gb
Video: S3 Savage3D 8 MB Monitor 17"
Sound card pe placa de bază: ESS TM ES1938S Solo-1
CD-ROM: LITEON LTN382 40X
Tastatură: Qtronix multimedia Scorpius-98N
Mouse: Qtronix Lynx 3D
Modem: HSF 56k date/fax/voce intern, cu cipset Rockwell
Boxe: Mecmar LS-26A, 135W PMPO

UMAX Astra 1220U
rezoluție optică 600x1200dpi
max. 9600x9600, col. 36bit, 1 pass

HEWLETT PACKARD
ROMÂNIA

HP DeskJet 610 C

SISTEC
THE GLOBAL SOLUTION
SCS® - STAR™

Placa de bază ATX, INTEL 810, 4 MB AGP&SOUND, UDMA 66
Procesor INTEL CELERON 400/128bit PPGA • Memorie 32 MB SDRAM, PC-10
Video 4 MB AGP ON BOARD • Hard-disk QUANTUM 6.4 GB UDMA
Tastatură GENIUS WIN 95 • Floppy 1.44 MB FDD • Mouse GENIUS SCROLL
Monitor 14" SVGA LR NI DIGITAL MPRII • CD-ROM
Sound ON BOARD AC 97 AUDIO
Boxe 120 W, ACTIVE SPEAKERS

ABONEAZĂ-TE
și câștigă!

Abonează-te
la PCMagazine România,
PC Report sau Gazeta de informatică,
reviste editate de Computer Press Agora, până la data de 15 decembrie 1999.
Vei participa la tragerea la sorți care va desemna câștigătorii premiilor.
Trimite cartile poștale din aceste reviste pe adresa:
Computer Press Agora, str.T.Vladimirescu nr.63/1, Tg.Mureș 4300 (CP 230/1)
la: tel. 065-166516, fax 065-166290

Începând cu acest număr...

CONCURS

... de programare

... la care pot participa elevi și studenți care doresc să-și pună în valoare cunoștințele în domeniul programării.

Concursul se va desfășura pe parcursul a 6(+1) runde, la fiecare rundă concurenții vor primi trei probleme. Problemele vor fi propuse de redacția GInfo în colaborare cu profesori și studenți implicați în pregatirea elevilor români care participă la diferite concursuri naționale și internaționale de informatică.

Regulament

Rezolvările problemelor vor fi trimise prin e-mail la adresa clara@cs.ubbcluj.ro, mesajul va trebui să aibă subiectul "Concurs GInfo". Fișierele atașate vor avea denumirea `CRxPy.ext` unde `x` reprezintă numărul rundei, `y` numărul problemei, iar `ext` este extensia specifică limbajului de programare folosit (valorile admise sunt `pas`, `cpp`, `c` și `java`). În cadrul mesajului va trebui să specificați numele, adresa și numărul dumneavoastră de telefon. Fișierele sursă vor fi compilate folosind compilatoarele *Borland Pascal 7.0*, *Borland C++ 3.1*, respectiv *JDK 1.2.1*; nu sunt acceptate opțiuni de compilare speciale, sursele vor fi compilate utilizând linii de comandă de tipul: `bpc nume`, `bcc nume`, respectiv `javac nume`. Nu sunt acceptate mai multe versiuni de rezolvare pentru o anumită problemă. Redacția își rezervă dreptul de a descalifica pe acei concurenți care vor trimite soluții identice.

În fiecare număr GInfo va apărea clasamentul intermediar din ziua închiderii ediției.

Bursele Agora

Primii zece clasați la sfârșitul celor șase etape vor participa la etapa finală care se va desfășura "în timp real" la Liceul de Informatică "Tiberiu Popoviciu" din Cluj.

Cel mai bine clasat student sau viitor student (elev în clasa a XII-a) va primi o bursă Agora pe perioada unui an universitar. Valoarea bursei va fi de aproximativ 50$ pe lună.

Cel mai bine clasat elev va primi și el o bursă Agora pe perioada unui an școlar, valoarea acesteia fiind de aproximativ 30$ pe lună.

Termen

Rezolvările problemelor publicate în acest număr pot fi trimise până la data de:

15 Noiembrie 1999

Runda 1

CR1P1 Volumul piramidei

Dându-se coordonatele a patru puncte din spațiu să se determine volumul tetraedrului determinat de cele 4 puncte. Datele de intrare se vor citi din fișierul **VOLUM.IN** care conține pe patru linii câte trei numere reale reprezentând coordonatele celor patru puncte. Volumul tetraedrului (cu o aproximație de 0.001) va fi scris în fișierul **VOLUM.OUT**. În cazul în care cele patru puncte nu pot determina un tetraedru fișierul de ieșire va conține numai numărul 0. Programul va trebui să furnizeze rezultatul în cel mult o secundă.

Exemplu

VOLUM.IN	VOLUM.OUT
0 0 0 1 0 0 0 1 0 0 0 1	0.167

CR1P2 Dansul fluturilor

Directorul circului Hilbert s-a hotărât să adauge un nou act la spectacol și a ales dansul fluturilor care este dansat, simultan, de n fluturi numerotați de la 1 la n. Inițial fluturii se află în câmpuri numerotate de la 1 la n. Există n săgeți desenate în câmpuri astfel încât din fiecare câmp pornește exact o săgeată și în fiecare câmp ajunge exact o săgeată.

La început fiecare fluture se află în câmpul cu numărul corespunzător (al k-lea fluture se află în câmpul k). La fiecare secundă, fiecare fluture sare din câmpul în care se află în câmpul indicat de săgeata ce pornește din acel câmp (toți fluturii sar deodată). Dansul fluturilor se termină când toți fluturii se află din nou în pozițiile din care au plecat. Directorul vrea ca spectacolul să dureze cât mai mult posibil, deci trebuie să realizeze o aranjare inteligentă a săgeților.

Scrieți un program care citește numărul n al fluturilor și determină o aranjare a săgeților astfel încât spectacolul să dureze cât mai mult timp posibil. Programul trebuie să determine și durata spectacolului (T).

Din fișierul **FLUTURI.IN** se va citi numărul n al fluturilor (1<n<1000), iar în fișierul **FLUTURI.OUT** se va scrie timpul T (în secunde). Programul va trebui să furnizeze rezultatul în cel mult o secundă.

Exemplu

FLUTURI.IN	FLUTURI.OUT
8	15

CR1P3 Hambar

Farmer John dorește să amplaseze un hambar mare de formă pătrată pe terenul de formă pătrată al fermei sale. Deoarece nu-i place să taie copaci la ferma sa, vrea să găsească o locație pentru hambarul său care să-i permită să-l construiască pe un teren fără copaci. Pentru scopurile noastre, terenul lui a fost împărțit în N*N parcele. Intrarea conține o listă cu parcelele care conțin copaci. Sarcina voastră este de a determina cel mai mare hambar de formă pătrată care poate fi amplasat pe teren fără să fie nevoie ca unii copaci să fie tăiați. Laturile hambarului trebuie să fie paralele cu axa orizontală, respectiv cea verticală.

Exemplu

Să considerăm următoarea reprezentare a terenului lui Farmer John unde '.' reprezintă o parcelă care nu conține copaci, iar '#' o matrice care conține copaci.

```
    1 2 3 4 5 6 7 8
  1 . . . . . . . .
  2 . # . . . # . .
  3 . . . . . . . .
  4 . . . . . . . .
  5 . . . . . . . .
  6 . . # . . . . .
  7 . . . . . . . .
  8 . . . . . . . .
```

Cel mai mare hambar are dimensiunea 5*5 și poate fi amplasat în partea din dreapta-jos a matricei, în oricare dintre cele două moduri posibile.

Date de intrare

Datele se citesc din fișierul **HAMBAR.IN** care conține:

- pe prima linie două numere întregi N (1≤N≤200) și T (1≤T≤500) reprezentând numărul de parcele de pe o latură, respectiv numărul parcelelor pe care cresc copac;
- pe liniile 2..T+1 două numere întregi din intervalul [1,n] indicând linia și coloana unei parcele pe care cresc copaci.

Date de ieșire

Fișierul **HAMBAR.OUT** va conține o singură linie pe care se va scrie lungimea maximă a unei laturi a hambarului.

Exemplu

INPUT.TXT	OUTPUT.TXT
8 3 2 2 2 6 6 3	5

Timp maxim de execuție/test: 1 secundă

I. CONCURSURI

Prima tabără de pregătire

În perioada 26 iulie - 4 august 1999 a avut loc la Școala Națională de Gaz din Mediaș prima tabără de pregătire și selecție a celor 17 componenți ai Lotului Național de Informatică.

Scopul organizării acestei tabere a fost selectarea echipelor României care vor participa la Olimpiada Balcanică de Informatică (BOI '99), respectiv la Olimpiada de Informatică a Europei Centrale (CEOI '99).

Au fost patru zile de concurs, în fiecare zi elevii au rezolvat câte trei probleme într-un interval de patru ore. Primii patru clasați au reprezentat România la CEOI '99, iar următorii patru au format echipa României pentru BOI '99.

BOI '99

În perioada 22-26 august s-a desfășurat la Ioannina, în Grecia, a opta ediție a Balcaniadei de Informatică. Alături de țara gazdă au mai participat echipe din Albania, Bulgaria, Cipru, Iugoslavia, Macedonia, România și Turcia.

Au fost două zile de concurs, în fiecare zi concurenții au rezolvat câte trei probleme într-un interval de cinci ore. Echipa României a fost formată din ***Adrian Cârcu*** (Bistrița), ***Tudor Leu*** (Focșani), ***Daniel Oprean*** (Cugir) și ***Angel Proorocu*** (Ploiești).

Ca de obicei, concurenții români au avut o comportare meritorie, ei obținând o medalie de aur prin Adrian Cârcu, două medalii de argint prin Tudor Leu și Angel Proorocu și o medalie de bronz prin Daniel Oprean.

Păstrând tradiția, în clasamentele neoficiale pe puncte și pe medalii, România s-a clasat pe primul loc.

CEOI '99

În perioada 2-9 septembrie s-a desfășurat la Brno, în Cehia, a șaptea ediție a Olimpiadei de Informatică a Europei Centrale. Au participat echipe din Bosnia-Herțegovina, Cehia, Croația, Germania, Polonia, România, Slovacia, Slovenia, Statele Unite ale Americii și Ungaria.

La fel ca și la Balcaniadă, au fost două zile de concurs, în fiecare zi au fost propuse spre rezolvare trei probleme, iar timpul efectiv de lucru a fost de cinci ore. România a fost reprezentată de ***Bogdan Batog*** (București), ***Bogdan Dumitru*** (Ploiești), ***Mihai Pătrașcu*** (Craiova) și ***Radu Ștefan*** (Brașov).

A fost cea mai bună comportare a echipei României din istoria CEOI, echipa noastră obținând două medalii de aur prin Mihai Pătrașcu și Ștefan Radu și două medalii de argint prin Bogdan Batog și Bogdan Dumitru.

Pentru prima, dată după mulți ani România s-a clasat pe primul loc în clasamentele neoficiale pe puncte și pe medalii ale acestui concurs.

A doua tabără de pregătire

În perioada 20-25 septembrie 1999 are loc la Sinaia a doua tabără de pregătire și selecție a Lotului Național de Informatică la care participă primii 11 clasați în urma concursului de la Mediaș. Alături de cei opt concurenți care au reprezentat România la concursurile internaționale desfășurate în această vară mai sunt prezenți Andrei Vancea (Cluj-Napoca), Octavian Costache (Ploiești) și Ioana Ileană (Alba-Iulia)

Sunt trei zile de concurs cu trei probleme în fiecare zi, iar primii patru clasați vor participa la ***Olimpiada Internațională de Informatică*** (***IOI '99***) care se va desfășura în acest an la Antalya, în Turcia.

Datorită faptului că ediția pentru acest număr se închide înainte ca această tabără de pregătire să înceapă, respectiv, înainte de IOI '99, nu suntem în măsură să vă oferim rezultatele obținute la acest concurs. Le veți afla citind următorul număr *GInfo* în care veți găsi detalii despre comportarea României la a unsprezecea ediție a Olimpiadei Internaționale de Informatică.

Vom încheia această rubrică cu urarea tradițională: ***Succes!***

Athlon și în România

Firma bucureșteană SPOT DIGITAL oferă începând cu data de 15 septembrie 1999 procesoare Athlon 500, produse de AMD. Prețul unui astfel de procesor este de 260$ pentru reselleri, respectiv 280$ pentru end-useri și nu include TVA; pe piața americană un astfel de procesor poate fi cumpărat cu 249$.

Intel vs. Inprise

Procesoarele *Pentium II* și *Pentium III* produse de compania Intel nu sunt total compatibile cu mediul *Borland Pascal 7.0* al companiei *Inprise*. Mai exact, în momentul rulării unui program *Pascal* care folosește unit-ul `CRT` apare eroarea 200:

```
Division by zero.
```

Datorită faptului că acest lucru nu se întâmpla la celelalte procesoare din familia x86 produse de *Intel*, se poate spune că această companie nu-și respectă promisiunea de a păstra compatibilitatea cu versiunile anterioare ale produselor sale.

Eroarea apare chiar dacă nu sunt apelate funcții sau proceduri `CRT`, nici chiar programul

```
program Test;
uses Crt;
Begin
End.
```

neputând fi rulat cu succes.

Această incompatibilitate nu permite rularea majorității programelor Pascal scrise în mediul *Borland Pascal 7.0*, acestea devenind inutilizabile dacă folosiți un *Pentium II* sau un *Pentium III*. În momentul în care laboratoarele de informatică din școlile românești vor fi dotate cu procesoare din această generație (sperăm cât mai curând) profesorii de informatică vor întâmpina dificultăți în predarea limbajului de programare *Pasçal*, limbaj prezent încă în programa școlară.

Pe plan internațional această problemă a trecut practic neobservată, deoarece aproape nimeni nu mai folosește mediul *Borland Pascal*, ci se programează în medii vizuale.

Vă vindem totuși un "pont": dezactivați memoria cache și să sperăm că vă veți descurca.

Guvernul american caută alternative la *Microsoft*

Îngrijorate de securitatea și siguranța pe care o oferă softul de la *Microsoft*, oficialitățile americane doresc să diversifice tipurile de sisteme de operare cumpărate de guvern. *Consiliul Național de Securitate* american va crea în curând un nou departament prin care va evalua modul în care agențiile federale ar putea utiliza mai mult soft open-source care este disponibil gratuit. În anul 2000, guvernul Statelor Unite va cumpara soft în valoare de 2 miliarde de dolari.

Inprise lansează Borland Delphi 5

Având o mai bună integrare cu sistemul de operare *Windows* și o parte de *Web* mai dezvoltată, ultima versiune de *Delphi* lansată pe piață suportă *HTML 4* și *XML* (*eXtensible Markup Language*). Utilitarul de dezvoltare rapidă a aplicațiilor pentru *Windows* a fost lansat într-o perioadă critica pentru *Inprise*, când veniturile au avut o evoluție descendentă în ultimele trimestre. *Delphi 5 Enterprise* costă 2.499$, iar prețul pentru *Delphi 5 Professional* este de 799$, versiunea standard costând doar 99,95$. (Prețurile sunt valabile în Statele Unite.)

PRINCIPIUL inducției în INFORMATICĂ

Doru Popescu Anastasiu

În continuare ne propunem să prezentăm probleme care pot fi rezolvate prin inducție matematică și permit scrierea de algoritmi. Datorită procesului iterativ de generare a soluțiilor se obțin în general algoritmi liniari.

P1: Perechi de numere

Se dau n ($n \geq 1$) perechi de numere întregi $(a_1,b_1), (a_2,b_2), \ldots, (a_n,b_n)$. Să se determine două numere întregi c și d cu proprietatea:

$(a_1^2+b_1^2)(a_2^2+b_2^2)\ldots(a_n^2+b_n^2)=c^2+d^2$.

Exemplu

Pentru n=3 și perechile:

2 3
1 3
2 2

se va afișa:

c=28
d=16

Soluție

Construcția numerelor c și d se poate realiza prin inducție matematică.

Dacă n=1 atunci putem lua $c=a_1$ și $d=b_1$.

Dacă $(a_1^2+b_1^2)(a_2^2+b_2^2)\ldots(a_k^2+b_k^2)=e^2+f^2$, atunci putem lua $c=ea_{k+1}+fb_{k+1}$ și $d=eb_{k+1}-fa_{k+1}$ pentru a obține:

$(a_1^2+b_1^2)(a_2^2+b_2^2)\ldots(a_k^2+b_k^2)(a_{k+1}^2+b_{k+1}^2)=c^2+d^2$.

Listing: P1.PAS

```
program p1;
var a,b:array[1..500] of Longint;
    c,d,c1,d1,n,i:Longint;
Begin
  Write('n='); Readln(n);
  for i:=1 to n do
  begin
    Write('Dati perechea ',i);
    Readln(a[i],b[i])
  end;
  c:=a[1]; d:=b[1];
  for i:=2 to n do
  begin
    c1:=c*a[i]+d*b[i];
    d1:=Abs(c*b[i]-d*a[i]);
    c:=c1; d:=d1
  end;
  Writeln('c=',c);
  Writeln('d=',d)
End.
```

P2: Număr par

Se consideră un număr natural par m. Se cere să se determine două numere naturale x și y, astfel încât $m=(x+y)^2+3x+y$.

Exemplu

Pentru m=6 se va afișa:

x=0; y=2

Soluție

Determinarea numerelor x și y o vom face prin inducție matematică, notându-le cu x_n și y_n unde m=2n.

Dacă m=0, putem lua $x_0=y_0=0$. Dacă afirmația este adevărată pentru n număr natural, atunci avem:

$2n=(x_n+y_n)^2+3x_n+y_n$.

Pentru a găsi x_{n+1} și y_{n+1} cu proprietatea:

$2n+2=(x_{n+1}+y_{n+1})^2+3x_{n+1}+y_{n+1}$ (*),

distingem cazurile:

I. $y_n=0$, atunci putem lua $x_{n+1}=0$; $y_{n+1}=x_n+1$ și (*) este verificată.

II. $y_n=1$, atunci, pentru $x_{n+1}=x_n+1$ și $y_{n+1}=y_n-1$, obținem (*).

Listing: P2.PAS

```
program p2;
var m,x,y,x1,y1,i:Longint;
```

```
Begin
  Write('Introduceti un numar natural par ');
  Readln(m);
  m:=m div 2;
  x:=0; y:=0;
  for i:=1 to m do
  begin
    if y=0 then begin x1:=0; y1:=x+1 end
           else begin x1:=x+1; y1:=y-1 end;
  x:=x1; y:=y1
  end;
  Writeln('x=',x); Writeln('y=',y)
End.
```

P3: Soluția unei ecuații

Pentru n număr natural, (n≥6). Să se determine o soluție în mulțimea numerelor naturale a ecuației:

$1/x_1^2 + 1/x_2^2 + \ldots + 1/x_n^2 = 1$.

Exemplu

Pentru n=7 o soluție este:

2 2 2 4 4 4 4

Soluție

Soluția problemei poate fi construită inductiv cu pasul 3, astfel:

- pentru n=6, avem:
  $1/2^2+1/2^2+1/2^2+1/3^2+1/3^2+1/6^2=1$;
- pentru n=7, avem:
  $1/2^2+1/2^2+1/2^2+1/4^2+1/4^2+1/4^2+1/4^2=1$;
- pentru n=8, avem:
  $1/2^2+1/2^2+1/2^2+1/3^2+1/3^2+1/7^2+1/14^2+1/21^2=1$.

Dacă pentru n avem:

$1/x_1^2+1/x_2^2+\ldots+1/x_n^2=1$, atunci pentru n+3, avem:

$1/x_1^2+1/x_2^2+\ldots+1/x_{n-1}^2+1/(4x_n^2)+1/(4x_n^2)+1/(4x_n^2)+1/(4x_n^2)=1$,

și deci soluția este

$(x_1, x_2, \ldots, x_{n-1}, 2x_n, 2x_n, 2x_n, 2x_n)$.

Listing: P3.PAS

```
program p3;
type vector=array[1..100]of Longint;
var x,y,z:vector;
    temp:Longint;
    i,k,n,rest,dimx,dimy,dimz:Integer;

procedure afisare(v:vector; dim:Integer);
var j:Integer;
begin
  for j:=1 to dim do Write(v[j],' ');
  Writeln
end;

Begin
  Write('n='); Readln(n); dimx:=6;
  x[1]:=2; x[2]:=2; x[3]:=2;
  x[4]:=3; x[5]:=3; x[6]:=6;
  for i:=1 to 3 do y[i]:=2;
  for i:=4 to 7 do y[i]:=4;
  dimy:=7; dimz:=8;
  for i:=1 to 3 do z[i]:=2;
  z[4]:=3; z[5]:=3;
  z[6]:=7; z[7]:=14; z[8]:=21;
  for i:=9 to n do
  begin
    rest:=i mod 3;
    case rest of
    0:begin
        temp:=x[dimx]; dimx:=dimx+3;
        for k:=0 to 3 do x[dimx+k-3]:=2*temp
      end;
    1:begin
        temp:=y[dimy]; dimy:=dimy+3;
        for k:=0 to 3 do y[dimy-3+k]:=2*temp
      end;
    2:begin
        temp:=z[dimz]; dimz:=dimz+3;
        for k:=0 to 3 do z[dimz-3+k]:=2*temp
      end
    end
  end;
  rest:=n mod 3;
  case rest of
  0: afisare(x,dimx);
  1: afisare(y,dimy);
  2: afisare(z,dimz)
  end
End.
```

P4: O împărțire a unui poligon convex

(propusă la ONI '99, clasele XI-XII)

Enunțul problemei a fost publicat în numărul 5 al GInfo.

Soluție

a) Se observă că numărul vârfurilor unui poligon cu centru de simetrie este par. Verificarea existenței punctului de simetrie se reduce la a arăta că laturile opuse două câte două sunt paralele și congruente.

b) Determinarea descompunerii se poate face prin inducție matematică astfel:

- dacă n este 4 atunci poligonul este chiar un paralelogram;
- pentru un poligon $A_1A_2\ldots A_n$ alegem două laturi opuse paralele, spre exemplu A_1A_2 și $A_{n/2+2}A_{n/2+1}$ și translatăm spre stânga pe direcția A_1A_2 cu lungimea A_1A_2 linia poligonală $A_{n/2+2}A_{n/2+3}\ldots A_nA_1$ obținând n/2-1 paralelograme și un poligon convex cu n-2 vârfuri, care moștenește proprietatea de a avea centru de simetrie. Astfel problema a fost redusă la împărțirea unui poligon cu n-2 vârfuri.

Listing: P4.PAS

```
program partitie_poligon;
type punct=record x,y:Integer end;
     sir=array[1..101] of punct;
```

```
var a,b:sir; n,i:Integer; f:Text;

function paralele(p1,p2,q1,q2:punct):Boolean;
var sw:Boolean; m1,m2:Real;
begin
  if p1.x=p2.x
  then sw:=q1.x=q2.x
  else
    if q1.x=q2.x then sw:=false
    else
    begin
      m1:=(p1.y-p2.y)/(p1.x-p2.x);
      m2:=(q1.y-q2.y)/(q1.x-q2.x);
      sw:=m1=m2
    end;
  paralele:=sw
end;

function seg_eg(p1,p2,q1,q2:punct):Boolean;
var l1,l2:Longint;
begin
  l1:=sqr(p1.x-p2.x)+sqr(p1.y-p2.y);
  l2:=sqr(q1.x-q2.x)+sqr(q1.y-q2.y);
  seg_eg:=l1=l2
end;

function centru_sim:Boolean;
var sw:Boolean;
begin
  if n mod 2=1
  then sw:=false
  else
  begin
    sw:=true;
    for i:=1 to n div 2 do
    begin
      if not paralele(a[i],a[i+1],
                a[n div 2 +i],a[n div 2+i+1])
      then sw:=false;
      if not seg_eg(a[i],a[i+1],
                a[n div 2 +i],a[n div 2+i+1])
      then sw:= false
    end
  end;
  centru_sim:=sw
end;

procedure impartire(n:Integer; a:sir);
var j,k:Integer;
begin
  if n=4 then
  begin
    Write(f,a[1].x,' ',a[1].y,' ',a[2].x);
    Write(' ',a[2].y,' ',f,a[3].x,' ');
    Write(a[3].y,' ',a[4].x,' ',a[4].y);
  end
  else
  begin
    j:=n div 2 +2;
    while j<=n do
    begin
      Write(f,a[j].x,' ',a[j].y,' ');
      Write(a[j+1].x,' ',a[j+1].y,' ');
      Write(f,a[j+1].x+a[2].x-a[1].x,' ');
      Write(a[j+1].y+a[2].y-a[1].y,' ');
      Write(f,a[j].x+a[2].x-a[1].x,' ');
      Writen(a[j].y+a[2].y-a[1].y);
      j:=j+1
    end;
    for j:=1 to n div 2 do b[j]:=a[j+1];
    k:=n div 2;
    for j:=n div 2 +3 to n do
    begin
      k:=k+1;
      b[k].x:=a[j].x+a[2].x-a[1].x;
      b[k].y:=a[j].y+a[2].y-a[1].y
    end;
    for i:=1 to k do a[i]:=b[i];
    a[k+1]:=a[1];
    impartire(k,a)
  end
end;

Begin
  Assign(f,'POLIGON.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do Readln(f,a[i].x,a[i].y);
  a[n+1]:=a[1]; Close(f);
  Assign(f,'POLIGON.OUT'); Rewrite(f);
  if not centru_sim then Write(f,'NU')
  else
  begin Writeln(f,'DA'); impartire(n,a) end;
  Close(f)
End.
```

Bibliografie

1. Titu Andreescu - *O extindere a unei probleme de teoria numerelor*, Gazeta Matematică Nr. 1/1980;
2. Constantin Cocea - *Două teoreme de reprezentare de tip Erdos-Suranyi*, Revista de Matematică din Timișoara, Nr. 1/1998;
3. Ion Cucurezeanu - *Probleme de aritmetică și teoria numerelor*, Ed. Tehnică 1976;
4. L. Panaitopol, etc. - *Inducția Matematică*, Editura GIL, Zalău, 1997;
5. Doru Popescu Anastasiu - *Principiul inducției în informatică*, Gazeta de Informatică 3/1998;
6. W. Sierpinski - *Elementary Theory of Numbers*, Warszawa, 1987.

Domnul Doru Popescu Atanasiu este cadru didactic la liceul "Radu Greceanu", Slatina. Poate fi contactat prin intermediul redacției.

Principiul cutiei lui Dirichlet

Mihai Oltean

În Gazeta de Informatică nr. 9-10 din 1994 domnul prof. dr. Horia Georgescu a prezentat un articol în care a încercat să atragă atenția rezolvitorilor de probleme asupra principiului enunțat de renumitul matematician Peter Gustav Lejeune Dirichlet (1805-1859). Înainte de a continua analiza unor probleme a căror rezolvare se bazează pe acest principiu, îl reamintim:

*"Se consideră M obiecte care trebuie plasate în N cutii și un număr natural K având proprietatea M>K*N. Atunci pentru oricare amplasament există o cutie care va conține cel puțin K+1 obiecte."*

1. Grade egale

Să se determine două submulțimi disjuncte ale mulțimii nodurilor unui graf neorientat cu proprietatea că suma gradelor nodurilor dintr-o submulțime este egală cu suma gradelor nodurilor din cealaltă submulțime.

Soluție

Vom arăta că există două astfel de submulțimi care au fiecare cardinalul 1, adică sunt formate dintr-un singur element. Cu alte cuvinte trebuie demonstrat că într-un graf neorientat există două noduri care au același grad. Valorile gradelor nodurilor sunt elemente din mulțimea {0,...,n-1}, care este o mulțime de cardinal n. Însă, dacă există un nod de grad n-1 înseamnă că el este conectat cu toate celelalte noduri, deci nu vor exista noduri izolate (de grad 0). Asta înseamnă că mulțimea gradelor nodurilor unui graf neorientat are cel mult n-1 valori distincte, care vor trebui distribuite la cele n noduri ale grafului, deci conform pricipiului cutiei, cel puțin două noduri vor avea același grad.

Referitor la implementarea practică a acestei probleme, ne interesează să găsim un algoritm de complexitate liniară care să determine cele două noduri în cauză. Un astfel de algoritm este asemănător cu algoritmul de sortare liniară, și anume: definim un vector:

```
b:array[0..MaxN] of Word;
```

în care pe poziția k reținem numărul nodurilor având gradul k. Acest vector se poate construi printr-o singură parcurgere a șirului gradelor nodurilor. Apoi se determină care grad k are atașat cel puțin două noduri (b[k] este cel puțin 2). Algoritmul este următorul:

Listing GRADE.PAS

```
program DouaGradeEgale;
const MaxN = 100;
var grad:array[1..MaxN] of 0..MaxN-1;
    b:array[0..MaxN-1] of 0..MaxN;
    nod:array[0..MaxN-1] of 1..MaxN;
    n,i,j:Word;

procedure Citire;
var f:Text; c:Char;
begin
  Assign(f,'GRAFD.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do
  begin
    for j:=1 to n do
    begin
      Read(f,c);
      Inc(grad[i],Ord(c)-48);
      Inc(grad[j],Ord(c)-48)
    end;
    Readln(f)
  end;
  Close(f)
end;
```

```
Begin
  Citire;
  for i:=1 to n do
  begin
    Inc(b[grad[i]]);
    if b[grad[i]]=2
    then
    begin
      Write('Nodurile cautate sunt: ');
      Writeln(nod[grad[i]],' si ',i);
      Exit
    end
    else nod[grad[i]]:=i
  end
End.
```

Un enunț echivalent este: "Într-o cameră se află n persoane, fiecare cunoscând un anumit număr de alte persoane (din aceea cameră). Să se arate că există două persoane care au același număr de cunoștințe".

Pentru un graf neorientat soluția de mai sus nu mai este valabilă. Pentru a rămâne totuși valabilă această rezolvare, putem impune condiția ca din fiecare nod să plece cel puțin un arc. În acest caz vom avea cel puțin două noduri cu același grad exterior.

2. Concurs

La un concurs de informatică au participat M elevi care au avut de rezolvat N probleme. Se cunoaște numărul x_k al elevilor care au reușit să rezolve problema k. Să se determine cel mai mare număr S astfel încât să existe sigur cel puțin un concurent care a rezolvat minim S probleme și să se determine și numărul celor care au rezolvat cel puțin S probleme.

Soluție

În total s-au rezolvat $S=x_1+x_2+...+x_N$ probleme. Dacă toți elevii au rezolvat cel puțin o problemă, atunci mai rămân S-M probleme. Raționamentul se continuă pentru două probleme, trei probleme etc., atât timp cât se mai poate scădea din S. Să presupunem că am ajuns la un k (deci elevii au rezolvat câte k probleme fiecare) pentru care S-kM este mai mic sau egal decât M. Numărul k+1 este sigur cel mai mare număr de probleme pe care le-a rezolvat un elev, iar S-kM este numărul de elevi despre care putem afirma cu siguranță că au rezolvat k+1 probleme. Toți acești pași se reduc la următorii doi:

1) Numărul maxim sigur de probleme rezolvate de către un elev este:
   S **div** M;
2) Numărul de elevi care îndeplinesc acest maxim este:
   S **mod** M.

3. Divizor

Se dă un număr impar a. Se cere să se determine un număr natural b, astfel încât 2^b-1 se divide la a.

Soluție

Considerăm numerele $2^0-1, 2^1-1, ..., 2^a-1$. Acestea sunt a+1 numere. Conform principiului cutiei cel puțin două dintre ele dau același rest prin împărțirea la a. Fie:

2^k-1 și 2^m-1 (k<m)

aceste numere. Atunci numărul $2^m-1-(2^k-1)=2^k(2^{m-k}-1)$ se divide prin a și deoarece a este impar, rezultă că $2^{m-k}-1$ se divide la a.

Deoarece avem certitudinea că există întotdeauna un astfel de multiplu, algoritmul de rezolvare este simplu și constă în a calcula resturile împărțirii lui 2^k-1 la a (pentru k=0...a).

Listing MULTIPLU.PAS

```
program multiplu;
const MaxN=30000;
var a:array[0..MaxN] of Word;
    n,i:word;

Begin
  repeat
    Write('n:'); Readln(n)
  until Odd(n);
  a[0]:=1; i:=0;
  repeat
    Inc(i);
    a[i]:=2*a[i-1] mod n
  until a[i]-1=0;
  Writeln(i)
End.
```

Un enunț echivalent este următorul: "Se dă un număr impar a. Se cere să se determine un multiplu al său care, scris în baza 2 (doi), are doar cifre de 1". Echivalența acestor două enunțuri este adevărată, deoarece un număr de forma 2^b-1 se scrie în baza doi ca o succesiune de b-1 de 1. Evident, se exclude cazul în care a=1.

Se poate demonstra și afirmația mai puternică: dacă numerele naturale a și c sunt prime între ele, atunci există un număr natural b astfel încât c^b-1 se divide cu a.

Enunțul echivalent este următorul: "Se dau două numere naturale a și c prime între ele. Să se determine un multiplu a lui a, care scris în baza c are doar cifre de 1".

4. Eliminare cifre

Se consideră un număr natural care are N cifre toate nenule. Se cere să se determine un alt număr natural format doar din cifrele numărului inițial (nu este obligatorie utilizarea tuturor cifrelor și în plus se poate folosi și cifra 0) și care se divide la N.

Soluție

Fie $A=\overline{a_1a_2\ldots a_N}$ numărul dat. Considerăm numerele:

$$\overline{a_N},\overline{a_{N-1}a_N},\ldots,\overline{a_1a_2\ldots a_N}$$

Acestea au resturile împărțirii la N în domeniul 0...N-1. Dacă există un rest 0, atunci numărul respectiv este cel căutat. Dacă nu, avem de a face cu N resturi din domeniul 1..N-1, adică N-1 resturi și conform principiului cutiei vor exista două resturi egale, deci două numere B și C care au același rest la împărțirea cu N. Scădem aceste numere și rezultatul este ceea ce ne interesează.

5. Suma a două elemente

Se consideră N numere naturale distincte, strict pozitive, strict mai mici decât 2N și toate diferite de N. Să se determine două dintre ele care au suma egală cu 2N.

Soluție

Dacă $0<a<2n$, atunci și $0<2n-a<2n$. Fie $a_1, a_2, \ldots, a_N$ cele N numere. Atunci și $2N-a_1$, $2N-a_2$,...,$2N-a_N$ îndeplinesc condițiile din enunț. Considerăm numerele $a_1, a_2, \ldots, a_N$, $2N-a_1, 2N-a_2, \ldots, 2N-a_N$. Ele sunt în număr de 2N. Dar în intervalul (0,2N) există doar 2N-1 numere distincte, deci conform principiului cutiei, cel puțin două dintre ele sunt egale, și anume unul din mulțimea $\{a_1, a_2, \ldots, a_N\}$ iar celălalt din mulțimea $\{2N-a_1, 2N-a_2, \ldots, 2N-a_N\}$. Fie ele a_i respectiv $2N-a_j$. Deci avem $a_i=2N-a_j$, adică $a_i+a_j=2N$. Cazul i=j este exclus prin enunț.

Acum, pentru că avem certitudinea existenței soluției putem aplica același algoritm ca și la problema 1.

6. Submulțimi

Fie $M=\{1,2,\ldots,N\}$ mulțimea primelor N numere naturale. Să se determine condiția pentru existența a două submulțimi A și B ale lui M, distincte, de cardinal mai mic sau egal cu K astfel încât suma numerelor dintr-o submulțime să fie egală cu suma numerelor din cealaltă submulțime.

Soluție

Numărul submulțimilor de cardinal cel mult K este 2^K. Din acest număr scădem 1 (reprezentat de mulțimea vidă).

Suma numerelor unei submulțimi $A \subset M$ de cardinal cel mult K este cuprinsă între 1 (mulțimea formată doar din elementul 1) și $S=N+(N-1)+\ldots(N-K+1)$, adică în total S valori distincte pentru suma elementelor submulțimilor. Pentru ca să existe cel puțin două submulțimi egale, trebuie să se îndeplinească condiția $2^K-1>S$, inegalitate care reprezintă condiția care ne interesează.

7. Sumă submulțime

Se consideră N numere naturale nenule a căror sumă este strict mai mică decât 2N. Să se determine o submulțime de sumă N.

Soluție

Condiția ca suma numerelor să fie strict mai mică decât 2N este necesară, deoarece în caz de egalitate s-ar putea să nu avem soluție. Spre exemplu, dacă N=2 și numerele sunt 1 și 3, nu găsim nici o submulțime de sumă 2.

Fie $a_1, a_2, \ldots, a_N$ cele N numere. Calculăm sumele parțiale $a_1, a_1+a_2, a_1+a_2+a_3, \ldots, a_1+a_2+\ldots+a_N$. Resturile împărțirii acestor numere la N sunt în domeniul $\{0,\ldots,N-1\}$. Dacă unul din resturi este 0, atunci suma care a generat acel rest este ceea ce căutăm. În caz contrar, conform principiului cutiei, avem două resturi egale (care se termină prin a_i și a_j, i<j) și atunci submulțimea căutată este $a_{i+1}, \ldots, a_j$. Faptul că suma tuturor elementelor este strict mai mică decât 2N ne asigură că submulțimea obținută are suma exact N și nu un multiplu de N.

8. Produs submulțime

Se consideră N numere naturale. Se cere să se determine o submulțime a lor astfel încât produsul elementelor din submulțime să fie multiplu de N.

Soluție

Aplicăm aceleași raționamente ca și în cazul în care avem de a face cu sumă în loc de produs.

Listing SIR.PAS

```
program sir;
const MaxN = 1000;
var a:array[1..MaxN] of Longint;
    rest:array[0..MaxN] of Longint;
    n,i,j:Word;
    f:Text;

Begin
  Assign(f,'SIR.IN');
  Reset(f);
  rest[0]:= 0;
  for i:= 1 to n do
  begin
    Read(f,a[i]);
    rest[i]:=rest[i-1]*a[i] mod n;
    if rest[i]=0
    then
    begin
      Write('Secventa de la 1 la ',i);
      Writeln('este o solutie a problemei');
      Close(f);
      Exit
    end
  end;
  Close(f);
  for i:=1 to n-1 do
    for j:=i+1 to n do
      if rest[i]=rest[j]
      then
```

```
    begin
      Write('Secventa de la ',i+1,' la ');
      Writeln(j,'e o solutie a problemei');
      Exit
    end
End.
```

Observații:
Cazurile în care se cere ca submulțimea să aibă un număr minim de elemente, sau multiplul obținut să fie minim se rezolvă prin alte metode.

9. Produse divizibile

Se consideră o mulțime A de N+1 numere naturale nenule, mai mici sau egale cu 2N. Să se determine două submulțimi X și Y, incluse în A, astfel încât produsul elementelor dintr-o submulțime să fie divizibil cu produsul elementelor din cealaltă submulțime.

Soluție
Vom arăta că în mulțimea A care îndeplinește condițiile din enunț există două elemente care se divid unul pe celălalt.

Fie $a_1, a_2, \ldots, a_{N+1}$ cele N+1 numere naturale. După o prealabilă ordonare și renumerotare avem:

$$1 \le a_2 \le a_2 \le \ldots \le a_{N+1} \le 2N.$$

Orice număr natural se poate scrie sub forma $2^x \cdot y$, unde x este un număr natural, iar y este un număr natural impar. Deci $a_1=2^{x_1}\cdot y_1$, $a_2=2^{x_2}\cdot y_2$,..., $a_{N+1}=2^{x_{N+1}}\cdot y_{N+1}$.

Dar între 1 și 2N există exact N numere impare și deci, conform principiului cutiei, două dintre numerele $y_1 y_2, \ldots, y_{N+1}$ sunt egale. Fie acestea y_i și y_j cu $i<j$. Atunci numărul $a_i=2^{x_i}\cdot y_i$ divide pe $a_j=2^{x_j}\cdot y_j$.

10. Progresie

Fie mulțimea M de cardinal N și a cărei termeni sunt în progresie aritmetică de rație A și cu primul termen X. Să se arate că oricare ar fi H o submulțime a lui M, de cardinal

```
(N - 2) div 2 + 3
```

există în ea două elemente a căror sumă este AN + 2X.

Soluție
Elementele mulțimii M sunt de forma:

```
X, X+A, X+2A, ..., X+(N-1)A.
```

Dacă N este par, atunci facem următoarea partiționare în (N - 2) **div** 2 + 2 clase:

```
{X}, {(AN+2X) div 2}, {X+A,X+(N-1)A},
{X+2A,X+(N-2)A}, ...
```

Dacă N este impar, atunci facem următoarea partiționare în (N-2) **div** 2+2 clase:

```
{X}, {X+A,X+(N-1)A}, {X+2A,X+(N-2)A}, ...
```

Deoarece fiecare partiționare are câte

```
(N - 2) div 2+2
```

clase, înseamnă că orice submulțime H cu proprietățile din enunț conține cel puțin un element din fiecare clasă. Deci va exista cel puțin o clasă care conține ambele elemente. Dar suma elementelor din clasa respectivă este exact ceea ce ne interesează.

Probleme Propuse:

11. Determinare submulțime

Se dau N numere naturale $a_1, a_2, \ldots, a_N$ strict pozitive a căror sumă este strict mai mică decât 2N-1. Să se determine o submulțime M a acestor numere astfel încât:

$$\sum_{x \in M} x + \sum_{i=1}^{N} a_i = 2N-1$$

12. Numere consecutive

Din oricare 10 numere consecutive putem alege unul prim cu toate celelalte.

13. Permutare

Fie $a_1, a_2, \ldots, a_N$ o permutare a numerelor $1, 2, \ldots, N$. Să se arate că produsul $(a_1-1)(a_2-2)\ldots(a_N-N)$ este par, dacă N este impar.

14. Sumă divizibilă

Să se arate că din 2N-1 numere naturale strict pozitive, există N a căror sumă este divizibilă cu N.

Soluție
Caz particular: Din oricare trei numere se pot alege două de sumă pară. Dacă printre cele trei numere se găsesc două pare, atunci suma lor va fi un număr par. În caz contrar vor exista două numere impare a căror sumă este un număr par. Aici cutiile au fost reprezentate de multiplii 2k, 2k+1.

O altă modalitate de a demonstra această afirmație se găsește în [3] și [4].

Bibliografie:

1. Horia Georgescu - *Principiul cutiei lui Dirichlet*, Gazeta de Informatică 9-10/94, Editura Libris, Cluj Napoca, 1994
2. Adrian Ghioca, Nicolae Teodorescu - *Culegere de probleme*, SSMR, București, 1987
3. *Probleme de matematică traduse din revista sovietică Kvant*, EDP, București, 1983
4. Oltean Mihai, *Numere prime și divizibilitate*, Gazeta de Informatică 2/99, Editura Computer PRESS Agora, Târgu Mureș, 1999
5. Colecția *Gazetei Matematice*

Mihai Oltean este student al Universității "Babeș Bolyai", Cluj. Poate fi contactat prin intermediul redacției.

Pagini WEB cu JavaScript

Mihai Scorțaru

Serialul despre proiectarea paginilor Web continuă cu prezentarea celui mai utilizat limbaj de scriptare folosit în acest moment. Folosirea limbajului JavaScript îmbunătățește calitatea paginilor dumneavoastră deoarece acestea pot deveni cu adevărat interactive. În primele episoade vom prezenta conceptele care stau la baza scrierii codului JavaScript, urmând ca apoi să prezentăm modalități prin care se pot crea pagini Web foarte interesante.

Prezentare generală

Firma *Netscape Communications*, din dorința de a extinde funcționalitatea popularului său program de navigare *Netscape Navigator*, a dezvoltat un limbaj de programare, astfel încât programele scrise în acest limbaj să poată fi incluse în cadrul paginilor Web. Inițial *Netscape* și-a denumit noul limbaj *LiveScript*, iar apoi, pentru a profita de renumele obținut de limbajul de programare *Java*, denumirea a fost schimbată în *JavaScript*.

În general, aplicațiile, indiferent cât de bine sunt realizate, nu pot satisface cerințele tuturor utilizatorilor, motiv pentru care acestora li se pune la dispoziție un limbaj simplu de programare cu ajutorul căruia pot să creeze extensii ale aplicației. De obicei, aceste miniprograme nu sunt compilate, ci interpretate linie cu linie în timpul rulării aplicației. Cel mai obișnuit exemplu de astfel de microaplicații îl reprezintă macro-urile din *Microsoft Office*. Aceste microaplicații poartă denumirea generică de ***script***-uri, iar colecția de comenzi și instrucțiuni care pot apărea în cadrul unui script poartă denumirea de ***limbaj de scriptare*** sau ***limbaj de tip script***.

JavaScript este un limbaj de scriptare a cărui sintaxă seamănă, într-o oarecare măsură, cu cea a limbajelor *C*, *C++*, *Pascal*, *Java* sau *Delphi*, dar care are anumite particularități care îl fac să se deosebească esențial de orice alt limbaj de programare.

Pentru ca programul de navigare să poată executa instrucțiuni *JavaScript*, el trebuie configurat în așa fel încât să nu ignore astfel de instrucțiuni. Totodată programul de navigare trebuie să recunoască instrucțiunile *JavaScript*; *Netscape Navigator* oferă suport pentru *JavaScript* începând cu versiunea *2.0*, în timp ce *Internet Explorer* oferă acest suport începând cu versiunea *3.0*.

Limbajul *JavaScript* este un limbaj bazat pe obiecte, adică un limbaj orientat obiect care conține obiecte predefinite.

De asemenea *JavaScript* este un limbaj proiectat astfel încât să reacționeze în momentul apariției unui eveniment.

Totodată *JavaScript* este un limbaj sigur în sensul că, în principiu, nu pot fi scrise microaplicații care să aibă acțiuni dăunătoare asupra calculatorului pe care este vizualizată pagina Web. De exemplu, nu va putea fi scris un script care să scaneze discul și să șteargă anumite fișiere. Totuși există posibilitatea păstrării anumitor parole introduse de către utilizator, ceea ce ar putea deranja pe unii dintre aceștia. Din această cauză majoritatea programelor de navigare oferă o opțiune a cărei activare nu permite executarea script-urilor.

În sfârșit, *JavaScript* este un limbaj independent de platforma pe care rulează, adică un același script va putea fi executat sub *Unix*, *MacIntosh* sau *Windows* fără a fi necesară prezența unui emulator sau efectuarea anumitor modificări.

Mulți informaticieni (chiar profesioniști) nu înțeleg deosebirile dintre *Java* și *JavaScript*. Java este un limbaj de programare orientat obiect, dezvoltat de firma *Sun Microsystems*, în timp ce *JavaScript* are o structură bazată pe obiecte și a fost dezvoltat de firma *Netscape Communications*. Limbajul *Java* poate fi folosit pentru proiectarea de aplicații independente sau pentru proiectarea de microaplicații care pot fi incluse în cadrul paginilor Web și care poartă denumirea de ***applet***-uri și sunt și ele independente de platformă. Iată câteva deosebiri esențiale între *Java* și *JavaScript*:

- Aplicațiile *Java* sunt compilate în fișiere binare care sunt apoi interpretate de către ***VJM*** (*Virtual Java Machine - Mașină Virtuală Java*), în timp ce codul *JavaScript* este transmis ca un text obișnuit și este interpretat.
- *Java* este orientat obiect în timp ce *JavaScript* este bazat pe obiecte.

- Codul *JavaScript* este inclus în cadrul documentului HTML; applet-urile *Java* sunt referite din cadrul unui document HTML, dar codul se află într-un fișier separat.
- Script-urile *JavaScript* sunt incluse prin intermediul marcajului <SCRIPT>, iar applet-urile Java prin intermediul marcajului <OBJECT>; există și marcajul <APPLET> dar folosirea lui nu mai este recomandată.
- În cadrul script-urilor *JavaScript* variabilele nu trebuie declarate, în timp ce în cadrul aplicațiilor *Java* variabilele trebuie declarate înainte de a fi folosite. Mai mult, în cadrul script-urilor *JavaScript*, o variabilă care este considerată ca având un anumit tip la un moment dat poate fi considerată ulterior ca având un alt tip; de exemplu, o variabilă poate fi folosită ca fiind un șir de caractere pentru ca apoi să fie considerată ca fiind un număr real.
- JavaScript folosește legarea dinamică, adică referințele sunt verificate în timpul rulării, în timp ce Java folosește legarea statică, adică referințele trebuie să existe în momentul compilării.

Mai există confuzii între *JavaScript* și *LiveWire* sau *LiveWire Pro*, acestea din urmă fiind tot produse *Netscape*. *LiveWire* nu este doar un limbaj de scriptare, cu toate că include un limbaj de scriptare compatibil cu *Java*. *LiveWire* include un editor/browser *WYSIWYG* (What You See Is What You Get — ceea ce vezi este ceea ce primești), un manager grafic pentru site-urile Web și (în versiunea *LiveWire Pro*) suport pentru baze de date *SQL* de la *Informix*, *Oracle*, *Sybase* și *Microsoft*. *LiveWire* va rula numai pe un server *Netscape*.

JavaScript este confundat uneori și cu limbajul *Pearl* care este un limbaj folosit în special pentru citirea, transmiterea și listarea conținutului unor fișier. Din motive de securitate, limbajul *JavaScript* nu permite citirea de fișiere text și nici transferul de date.

În general, limbajul *JavaScript* este folosit pentru efectuarea de calcule, citirea unor date dintr-o tabelă, proiectarea de ecrane HTML fără a folosi script-uri CGI.

Marcajul <SCRIPT>

În interiorul unui document HTML, instrucțiunile *JavaScript* sunt cuprinse, de obicei, în interiorul marcajului <SCRIPT>. Acest marcaj poate apărea atât în antetul documentului HTML (marcajul <HEAD>), cât și în corpul documentului (marcajul <BODY>). Script-urile definite în antet sunt încărcate înaintea încărcării restului paginii, antetul fiind un loc excelent pentru plasarea funcțiilor *JavaScript* pentru a fi siguri că acestea vor fi disponibile atunci când sunt apelate în alte secțiuni ale documentului HTML. Sintaxa marcajului <SCRIPT> este următoarea (în cazul folosirii limbajului *JavaScript*):

```
<SCRIPT language="JavaScript">
  //cod JavaScript
</SCRIPT>
```

Marcajul <SCRIPT> are un atribut suplimentar numit **src** cu ajutorul căruia puteți preciza URL-ul unui fișier care conține instrucțiunile *JavaScript* care formează script-ul respectiv. De obicei, fișierele care conțin astfel de instrucțiuni au extensia `.JS`.

Vom prezenta acum script-ul prin intermediul căruia poate fi afișat mesajul `Hello World!` în fereastra programului de navigare.

```
<HTML>
  <HEAD>
    <TITLE>
      Hello World!
    </TITLE>
    <SCRIPT language="JavaScript">
      document.write("Hello World!");
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

Datorită faptului că avem un document HTML al cărui corp este vid, ne-am fi așteptat ca pagina Web corespunzătoare să nu conțină nici un element. Acest lucru ar fi fost adevărat în cazul programelor de navigare care nu suportă limbajul *JavaScript*; totuși, în majoritatea cazurilor, programele de navigare vor executa instrucțiunea *JavaScript* `document.write()` care are ca efect plasarea în cadrul paginii Web a tuturor elementelor HTML care apar în cadrul parametrului. Parametrul ar fi putut conține chiar și marcaje HTML cum ar fi <BR>, <B> etc., efectele acestora fiind vizibile în pagină. Având în vedere cele spuse mai sus, documentul HTML prezentat este echivalent cu:

```
<HTML>
  <HEAD>
    <TITLE>
      Hello World!
    </TITLE>
  </HEAD>
  <BODY>
    Hello World!
  </BODY>
</HTML>
```

Limbajul *JavaScript* oferă multe alte facilități în afară de simpla inserare de elemente într-o pagină Web. Următorul exemplu ilustrează o modalitate de a cere vizitatorului să introducă anumite date:

```
<HTML>
  <HEAD>
    <TITLE>
      Salut!
    </TITLE>
    <SCRIPT language="JavaScript">
      var nume=window.prompt
   ("Vă rugăm să vă introduceți numele:","");
      document.write("Salut "+nume+"!");
    </SCRIPT>
  </HEAD>
</HTML>
```

La încărcarea paginii, comanda `window.prompt()` duce la apariția unei ferestre care conține o casetă text în interiorul căreia pot fi introduse date. Efectul comenzii din documentul HTML anterior poate fi observat în figura 1.

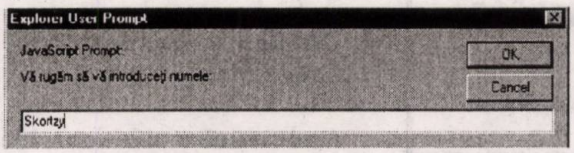


fig. 1 Obținerea datelor de la utilizator

Funcția `window.prompt()` returnează șirul de caractere introdus în cazul selectării butonului **`Ok`**, respectiv șirul *`null`* în cazul selectării butonului **`Cancel`** sau a celui de închidere. Fereastra nu poate fi redimensionată, iar încărcarea paginii Web are loc în background (fără efecte vizibile pe ecran) până în momentul închiderii acestei ferestre. În figurile 2 și 3 pot fi văzute efectele selectării butoanelor **`Ok`**, respectiv **`Cancel`**.

fig. 2 Efectul apăsării butonului **`Ok`**

fig. 3 Efectul apăsării butonului **`Cancel`**

Datorită faptului că nu toate programele de navigare suportă limbajul *JavaScript*, se recomandă ca instrucțiunile să fie cuprinse în cadrul unor comentarii HTML. Programele de navigare care nu recunosc marcajul **<SCRIPT>**, vor ignora instrucțiunile; celelalte programe de navigare vor ignora și ele textul dintre comentarii, dar îl vor transmite spre interpretorul *JavaScript*.

Evenimente

Așa cum am mai spus, limbajul *JavaScript* este bazat pe evenimente, acestea fiind folosite pentru a controla interacțiunea dintre utilizator și aplicație. Programele convenționale operează într-o manieră diferită, codul acestora fiind executat secvențial.

Pentru a specifica instrucțiunile *JavaScript* care trebuie executate la apariția unui eveniment, limbajul HTML pune la dispoziție anumite atribute pentru diferite marcaje, atribute care au ca valori instrucțiuni *JavaScript* (de obicei apeluri de funcții).

O primă categorie de evenimente *JavaScript* o reprezintă evenimentele sistem. Apariția acestor evenimente nu este o consecință directă a acțiunilor utilizatorului; ele pot apărea în momentul terminării încărcării paginii, în momentul în care o pagină începe să fie "descărcată", când a trecut un anumit interval de timp.

În momentul terminării încărcării unei pagini Web apare evenimentul *Load*. Atributul corespunzător acestui eveniment este **`onload`** și poate apărea numai în cazul marcajelor **<BODY>** sau **<FRAMESET>**. Când se lucrează cu cadre, uneori este necesar să ne asigurăm că anumite cadre au fost încărcate înainte ca altele să fie actualizate (de exemplu, am putea avea într-un cadru un formular ale cărui câmpuri sunt referite dintr-un alt cadru). Următorul exemplu ilustrează modul de folosire al atributului **`onload`**:

```
<HTML>
  <HEAD>
    <TITLE>
      Evenimentul Load
    </TITLE>
    <SCRIPT language="JavaScript">
      function Load(){
        window.alert("Hello World!");
      }
    </SCRIPT>
  </HEAD>
  <BODY onload="Load()">
    După terminarea încărcării acestei pagini
    va apărea o casetă de avertizare.
  </BODY>
</HTML>
```

După încărcarea paginii va apărea evenimentul *Load* ceea ce va duce la execuția funcției `Load()` funcție ce conține o singură comandă *JavaScript* care determină apariția casetei de avertizare din figura 4.

fig. 4 Casetă de avertizare

Un eveniment a cărui apariție poate fi folosită pentru "a face ordine în urma noastră" este *Unload*; acestui eveniment îi corespunde atributul HTML **onunload** care poate apărea tot în cazul marcajelor **<BODY>** și **<FRAMESET>**. Următorul exemplu ilustrează modul de folosire al acestui atribut:

```
<HTML>
  <HEAD>
    <TITLE>Evenimentul Unload</TITLE>
    <SCRIPT language="JavaScript">
      function Unload(){
        window.alert("La revedere!");
      }
    </SCRIPT>
  </HEAD>
  <BODY onunload="Unload()">
    Dacă încercați să închideți fereastra va
    apărea o casetă de avertizare.
  </BODY>
</HTML>
```

O altă categorie de evenimente o reprezintă evenimentele generate de acțiunea utilizatorului. În tabelul 1 sunt prezentate evenimentele care fac parte din această categorie.

Identificatori

La fel ca și în cazul majorității limbajelor de programare, în *JavaScript* fiecare variabilă, funcție etc. trebuie să aibă un identificator. Există și câteva restricții pentru identificatorii *JavaScript* și anume:

- Nu pot conține spații.
- Nu pot conține caracterul '-' deoarece acesta este interpretat ca fiind operatorul *minus*.
- Nu pot fi nici unul dintre cuvintele rezervate.

Așadar există o serie de cuvinte rezervate care nu pot fi folosite ca identificatori; lista acestor cuvinte este dată în tabelul 2.

Variabile

În *JavaScript* variabilele nu trebuie neapărat declarate; mai mult, există posibilitatea ca un anumit identificator să fie folosit pentru o variabilă de un anumit tip, pentru ca apoi el să fie folosit pentru o variabilă de alt tip. În momentul în care interpretorul întâlnește a variabilă nouă nedeclarată sau nefolosită anterior, el determină tipul variabilei în funcție de valoarea care este atribuită acelei variabile. Ca urmare, o variabilă întotdeauna trebuie inițializată pentru a nu apărea confuzii în momentul în care aceasta este folosită.

Deși această facilitate este exploatată de mulți programatori, deoarece îi scutește de timpul necesar declarării variabilelor, nu este recomandată folosirea ei, datorită faptului că foarte ușor pot apărea confuzii. Din aceste motive se recomandă declararea variabilelor înaintea folosirii lor.

La fel ca și în cazul variabilelor din majoritatea limbajelor de programare, în *JavaScript* acestea pot fi locale sau globale. În *JavaScript* există patru tipuri de valori pe care le poate

Evenimente determinate de utilizator

Eveniment	Atribut	Momentul apariției
Click	**onclick**	Efectuarea unui click în zona corespunzătoare elementului *pentru majoritatea elementelor*
DblClick	**ondblclick**	Efectuarea unui dublu-click în zona corespunzătoare elementului *pentru majoritatea elementelor*
MouseDown	**onmousedown**	Apăsarea butonului mouse-ului în zona corespunzătoare elementului *pentru majoritatea elementelor*
MouseUp	**onmouseup**	Eliberarea butonului mouse-ului în zona corespunzătoare elementului *pentru majoritatea elementelor*
MouseOver	**onmouseover**	Deplasarea cursorului mouse-ului în zona corespunzătoare elementului *pentru majoritatea elementelor*
MouseOut	**onmouseout**	Deplasarea cursorului mouse-ului în afara zonei corespunzătoare elementului *pentru majoritatea elementelor*
MouseMove	**onmousemove**	Deplasarea cursorului mouse-ului în cadrul zonei corespunzătoare elementului *pentru majoritatea elementelor*
Focus	**onfocus**	Primirea controlului de către element *numai pentru elementele din formulare*
Blur	**onblur**	Pierderea controlului de către element *numai pentru elementele din formulare*
KeyPress	**onkeypress**	Apăsarea și eliberarea unei taste *pentru majoritatea elementelor*
KeyDown	**onkeydown**	Apăsarea unei taste *pentru majoritatea elementelor*
KeyUp	**onkeyup**	Eliberarea unei taste *pentru majoritatea elementelor*
Submit	**onsubmit**	Transmiterea datelor dintr-un formular *numai pentru elementul* **<FORM>**
Reset	**onreset**	Ștergerea datelor dintr-un formular *numai pentru elementul* **<FORM>**
Select	**onselect**	Selectarea unui fragment de text *numai pentru elementele* **<INPUT>** *și* **<TEXTAREA>**
Change	**onchange**	Pierderea controlului de către element și schimbarea valorii corespunzătoare *numai pentru elementele* **<INPUT>**, **<SELECT>** *și* **<TEXTAREA>**

Tabelul 1

Cuvinte rezervate în JavaScript

abstract	const	final	import	null	super	try
boolean	continue	finally	in	package	switch	var
break	default	float	instanceof	private	synchronized	void
byte	do	for	int	protected	this	while
case	double	function	interface	public	throw	with
catch	else	goto	long	return	throws	
char	extends	if	native	short	transient	
class	false	implements	new	static	true	

Tabelul 2

lua o variabilă: numerice, logice, șiruri de caractere și "fără tip" (similar tipului `void` din *C/C++* sau *Java*).

În *JavaScript* șirurile de caractere sunt cuprinse între ghilimele sau apostrofuri. La fel ca și în limbajele *C/C++* și *Java*, există caractere care au codificare specială cum ar fi: *sfârșit de linie* (`\n`), *backspace* (`\b`), *form feed* (`\f`), *carrage return* (`\r`) sau *tab* (`\t`).

Expresii

O ***expresie*** este o colecție de constante, variabile și operatori a căror ordine respectă anumite reguli și care se evaluează la o anumită valoare care poate fi numerică, logică sau șir de caractere.

În *JavaScript*, practic, orice variabilă poate fi considerată logică deoarece orice număr diferit de zero poate fi considerat ca fiind valoarea logică `true`; de asemenea, orice șir de caractere nevid, diferit de `"null"`, poate fi considerat ca fiind valoarea logică `true`. Valorile 0, respectiv șir vid sau șir cu valoarea `"null"` pot fi considerate ca fiind valoarea logică `false`.

Identificatorii pentru variabile nu pot fi nici unul dintre cuvintele rezervate, nu pot conține spații și nu pot începe cu cifre, chiar dacă pot conține cifre în interior.

În *JavaScript* există trei tipuri de expresii:

- expresii în urma evaluării cărora se obține o valoare numerică (***expresii aritmetice***);
- expresii în urma evaluării cărora se obține un șir de caractere (***expresii șir de caractere***);
- expresii în urma evaluării cărora se obține o valoare logică (***expresii logice***).

Operatori și operanzi

Operatorii sunt simboluri speciale care controlează modul în care sunt evaluate expresiile; ***operanzii*** reprezintă restul componentelor expresiei (variabile, constante sau subexpresii).

Operatorii pot fi binari sau unari; un operator binar necesită prezența a doi operanzi, în timp ce un operator unar necesită prezența unui singur operand.

Operatorii folosiți în cadrul expresiilor aritmetici poartă denumirea de ***operatori aritmetici***. Operatorii aritmetici sunt `'+'` (adunare), `'-'` (scădere), `'*'` (înmulțire), `'/'` (împărțire), `'%'` (modulo), `'++'` (incrementare), `'--'` (decrementare) și din nou `'-'` (negație unară) având aceleași semnificații ca și în cazul limbajelor *C/C++* sau *Java*.

Operatorii folosiți pentru compararea unor valori poartă denumirea de ***operatori de comparare*** sau ***comparatori***. Comparatorii sunt `'=='` (egal), `'!='` (diferit), `'>'` (mai mare), `'>='` (mai mare sau egal), `'<'` (mai mic), `'<='` (mai mic sau egal); rezultatul este o valoare logică.

Șirurile de caractere pot fi concatenate, operatorul de concatenare al acestora fiind `'+'`.

În expresiile logice, alături de operatorii de comparare pot apărea și ***operatori logici***. Operatorii logici sunt `'&&'` (conjuncție), `'||'` (disjuncție) și `'!'` (negație).

În expresiile aritmetice pot apărea și ***operatori pe biți***. În JavaScript aceștia sunt `'&'` (ȘI), `'|'` (SAU), `'^'` (SAU exclusiv), `'~'` (NU); rezultatul este o valoare numerică.

O altă categorie de operatori o reprezintă ***operatorii de deplasare*** (***shift-are***). Pentru a deplasa biții unui număr cu un număr de poziții spre dreapta, operatorul corespunzător este `'<<'`. Pentru deplasare spre stânga există doi operatori și anume `'>>'` și `'>>>'`; diferența dintre cei doi operatori apare numai în cazul numerelor negative; în ca-

Operatori de atribuire

Operator	*Exemplu*	*Echivalență*
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y
<<=	x<<=y	x=x<<y
>>=	x>>=y	x=x>>y
>>>=	x>>>=y	x=x>>>y
&=	x&=y	x=x&y
^=	x^=y	x=x^y
|=	x|=y	x=x|y

Tabelul 3

zul operatorului '<<' în urma deplasării spre stânga, ultimii biți din reprezentare vor fi completați cu valoarea bitului semn ('1' pentru numere negative și '0' pentru numere pozitive), în timp ce în cazul operatorului '>>>' acești biți sunt completați cu '0' indiferent de valoarea bitului semn.

Probabil cea mai importantă categorie de operanzi o reprezintă *operatorii de atribuire*. Cel mai folosit operator de atribuire este '=', cu ajutorul căruia se atribuie primului operand valoarea celui de-al doilea operand; primul operand trebuie să fie întotdeauna o variabilă. Ceilalți operatori de atribuire au fost introduși pentru a permite o scriere prescurtată pentru expresii de genul 'x=x+y', expresie care poate fi scrisă 'x+=y'. În tabelul 3 se poate vedea lista operatorilor de atribuire prezenți în *JavaScript*, împreună cu un exemplu de utilizare și expresia echivalentă folosind operatorul '='.

La fel ca și în *C/C++* sau *Java* există și *operatorul condițional* '?:', *operatorul virgulă* ',', operatorul membru '[]' și *operatorul apel* '()'.

Ordinea operațiilor (precedența operatorilor)

În cazul anumitor expresii care conțin mai mulți operatori trebuie să existe o ordine în care sunt efectuate operațiile. Astfel operatorii primesc un anumit "grad de prioritate"; în cazul evaluării expresiei vor fi efectuate mai întâi operațiile datorate operatorilor cu cea mai mare prioritate, urmând ca apoi să se treacă la cele datorate operatorilor cu următorul grad de prioritate. În cazul în care se ajunge în situația ca într-o expresie să apară doi operanzi cu același grad de prioritate, operațiile vor fi efectuate de la stânga la dreapta. În tabelul 4 poate fi observată precedența operatorilor, prioritatea scăzând pe măsură ce operatorul apare mai jos în tabel.

Ordinea operațiilor

Apel și membru	() []
Negație, incrementare și decrementare	! ~ - ++ --
Înmulțire și împărțire	* / %
Adunare și scădere	+ -
Deplasare (shift-are)	<< >> >>>
Comparare	< <= > >=
Egalitate	== !=
ȘI pe biți	&
SAU exclusiv pe biți	^
SAU pe biți	|
Conjuncție	&&
Disjuncție	||
Condițional	?:
Atribuire	= += -= *= \= %= <<= >>= >>>= &= ^= |=
Virgulă	,

Tabelul 4

Instrucțiuni

O *instrucțiune* este o colecție de cuvinte cheie, operatori și operanzi, terminată cu caracterul ';' și care are o anumită semnificație pentru interpretorul *JavaScript*. Componentele instrucțiunii pot fi separate prin unul sau mai multe caractere albe; în JavaScript caracterele albe sunt spațiu, tab și sfârșit de linie.

Instrucțiunile *JavaScript* se împart în patru categorii: comentarii, instrucțiuni condiționale, instrucțiuni de ciclare și instrucțiuni pentru manipularea obiectelor.

Declararea variabilelor

În *JavaScript* variabilele sunt declarate cu ajutorul instrucțiunii var; identificatorii care apar după cuvântul rezervat var sunt considerați identificatori de variabile. Opțional există și posibilitatea inițializării variabilelor declarate în acest mod.

O variabilă declarată în interiorul unei funcții "există" (poate fi folosită) numai în cadrul acelei funcții, în timp ce o variabilă declarată în afara oricărei funcții poate fi folosită în cadrul documentului curent.

Comentarii

În *JavaScript* există aceleași tipuri de comentarii ca și în cazul limbajelor *C/C++* sau *Java*; textul care apare între '/*' și '*/' precum și textul de pe o linie care apare după '//' este considerat comentariu.

Instrucțiuni condiționale

Instrucțiunile condiționale permit controlarea, pe baza unei condiții, a modului în care sunt executate instrucțiunile *JavaScript*. În mod normal instrucțiunile sunt executate secvențial, dar cu ajutorul instrucțiunilor condiționale poate fi schimbată ordinea de execuție.

În *JavaScript* instrucțiunea condițională folosește cuvintele cheie if și else și are următoarea sintaxă:

```
if (condiție)
  {//instrucțiuni pentru condiție adevărată}
else {//instrucțiuni pentru condiție falsă}
```

Acoladele au un rol foarte important în *JavaScript*, cu ajutorul lor se pot grupa mai multe instrucțiuni într-un singur bloc de instrucțiuni care este tratat ca o singură instrucțiune.

Instrucțiuni de ciclare

Adeseori se dorește executarea repetată a mai multor instrucțiuni; acest lucru este posibil cu ajutorul instrucțiunilor de ciclare. În *JavaScript* există două tipuri de instrucțiuni de ciclare: for și while.

Sintaxa instrucțiunii `for` este:

`for` (*inițializare; condiție; incrementare*)
{//*corpul ciclului*}

Oricare dintre cele patru secțiuni (de inițializare, condiția, de incrementare sau corpul) poate lipsi. Execuția unei instrucțiuni `for` constă în următorii pași:

1. se execută secțiunea de inițializare;
2. se evaluează condiția;
3. în cazul în care condiția este adevărată se execută corpul ciclului; în caz contrar se termină execuția ciclului;
4. se execută secțiunea de incrementare;
5. se revine la pasul 2.

Sintaxa instrucțiunii `while` este:

`while` (*condiție*) {//*corpul ciclului*}

În cazul instrucțiunii `while` poate lipsi corpul ciclului, prezența condiției fiind obligatorie. Execuția unei instrucțiuni `while` constă în următorii pași:

1. se evaluează condiția;
2. în cazul în care condiția este adevărată se execută corpul ciclului; în caz contrar se termină execuția ciclului;
3. se revine la pasul 1.

Instrucțiunile break și continue

Instrucțiunea `break` permite ieșirea forțată dintr-o instrucțiune de ciclare; în momentul în care interpretorul *JavaScript* întâlnește o astfel de instrucțiune, el întrerupe execuția ciclului în interiorul căruia se află aceasta.

Instrucțiunea `continue` permite saltul peste anumite instrucțiuni din corpul ciclului care nu mai trebuie executate în anumite condiții. În cazul unui ciclu `for` se trece automat la pasul 4 al execuției ciclului, în timp ce în cazul unui ciclu `while` se trece la pasul 1.

Instrucțiuni pentru manipularea obiectelor

Pentru a ușura manipularea obiectelor, limbajul JavaScript pune la dispoziție instrucțiunile `for...in` și `with`, operatorul `new` și cuvântul cheie `this`.

În momentul creării unei variabile se creează, de fapt, o ***instanță*** a acelei variabile. La fel este și cazul obiectelor, dar, datorită faptului că acestea nu sunt simple variabile, este nevoie de un operator special pentru crearea unei instanțe a unui obiect. Pentru a crea o variabilă obiect sintaxa folosită este:

variabilă=`new` *tip_obiect(listă_parametri)*

Variabilă este identificatorul variabilei obiect, *tip_obiect* este tipul obiectului care este creat, iar *listă_parametri* este lista parametrilor necesari pentru ca obiectul să fie creat corect.

Obiectele sunt formate din ***proprietăți*** și ***metode***; datorită faptului că proprietățile pot fi la rândul lor obiecte (pot exista mai multe niveluri de imbricare) este destul de dificil să ne dăm seama unde anume ne aflăm în ierarhia de obiecte la un moment dat. Pentru a ușura sarcina programatorilor, limbajul *JavaScript* oferă o "scurtătură" pentru referirea obiectului curent și anume cuvântul cheie `this`. Acest cuvânt cheie se traduce prin "obiectul curent", indiferent de tipul acestui obiect.

Instrucțiunea `for...in` este o instrucțiune specială de ciclare care folosește toate proprietățile unui obiect. Sintaxa acestei instrucțiuni este:

`for` (*proprietate* `in` *obiect*)
{//*instrucțiuni*}

Această instrucțiune este deseori folosită pentru a afișa toate proprietățile unui anumit obiect. În următorul exemplu vom afișa toate proprietățile unui document:

```
<HTML>
  <HEAD>
    <TITLE>
      Proprietățile documentului
    </TITLE>
  </HEAD>
  <BODY>
    <H3 align="center">
      Proprietățile documentului</H3>
    <HR>
    <SCRIPT language="JavaScript">
      for (proprietate in document) {
        document.write(proprietate+" ");
      }
    </SCRIPT>
  </BODY>
</HTML>
```

În figura 5 este prezentată fereastra programului de navigare în momentul încărcării documentului HTML anterior.

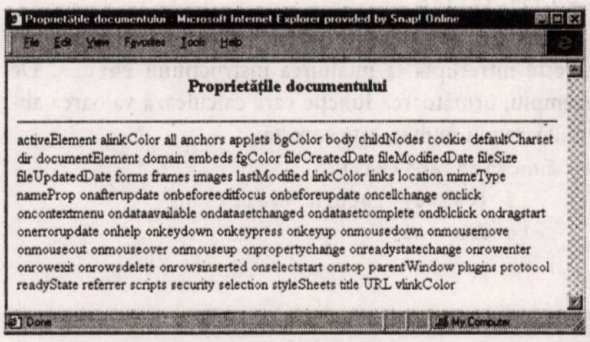


Fig. 5 Proprietățile obiectului `document`

Valoarea returnată în variabila `proprietate` este un șir de caractere; valoarea proprietății poate fi referită folosind o indexare specială a vectorilor:

`document[proprietate].`

Dacă cunoaștem proprietățile pe care dorim să le manipulăm și vrem să efectuăm operații asupra câtorva pro-

prietăți ale aceluiași obiect, poate fi folosită o prescurtare prin intermediul instrucțiunii `with`. Această instrucțiune are ca parametru un obiect și toate proprietățile la care se face referire în corpul acestei instrucțiuni sunt considerate a fi proprietățile obiectului transmis ca parametru. Sintaxa acestei instrucțiuni este:

```
with (obiect) {//corpul instrucțiunii}
```

Așadar în loc să scriem:

```
document.fgColor="#000000";
document.bgColor="#FFFFFF";
```

am putea scrie:

```
with (document){
  fgColor="#000000";
  bgColor="#FFFFFF";
}
```

Funcții

O colecție de instrucțiuni care efectuează o anumită operație pot fi grupate într-o ***funcție***. Sintaxa unei funcții este următoarea:

```
function funcție (listă_parametri)
  {//corpul funcției}
```

Parantezele sunt obligatorii chiar dacă lista parametrilor este vidă. O funcție poate fi apelată printr-o instrucțiune de genul: *funcție(parametri)*. În cazul în care funcția nu are parametri apelul va fi *funcție()*, prezența parantezelor fiind obligatorie și în acest caz.

Există și posibilitatea ca o funcție să returneze o valoare care să fie atribuită unei variabile. Un exemplu de funcție care calculează pătratul unui număr este:

```
function sqr(val){
  return val*val;
}
```

Un exemplu de utilizare a acestei funcții poate fi `rez=sqr(123)`; în acest caz variabila `rez` va primi valoarea `1232=15.129`.

Trebuie precizat faptul că execuția corpului unei funcții este întreruptă la întâlnirea instrucțiunii `return`. De exemplu, următoarea funcție care calculează valoarea absolută a unui număr, este corectă:

```
function abs(val){
  if (val<0) return -val;
  return val;
}
```

Obiecte

Obiectele sunt elementele fundamentale în cazul oricărui limbaj orientat obiect. Obiectele software păstrează proprietățile fundamentale ale obiectelor reale; orice obiect real are anumite ***caracteristici*** și un anumit ***comportament***. În cazul obiectelor software, caracteristicile sunt înlocuite de ***proprietăți***, care sunt anumite variabile care caracterizează obiectul, iar comportamentul este înlocuit de ***metode*** care sunt funcții (sau proceduri) care descriu modul în care se comportă obiectul. La un moment dat un obiect real se află într-o anumită ***stare*** care poate fi determinată de caracteristicile sale; starea obiectelor software poate fi determinată cu ajutorul valorilor proprietăților sale.

De obicei valorile proprietăților sunt ascunse, ele fiind disponibile numai prin intermediul metodelor. Această tehnică a ascunderii datelor poartă denumirea de ***încapsulare***.

Pentru a comunica între ele, obiectele folosesc ***mesaje***; în *JavaScript* un mesaj constă din trei părți:

- *numele* obiectului care primește mesajul (destinatarul mesajului);
- *metoda* care trebuie executată (conținutul mesajului);
- *parametrii* asupra cărora acționează metoda (instrucțiuni speciale).

Clase de obiecte

În lumea reală toate obiectele fac parte dintr-o anumită ***categorie***; noțiunea corespunzătoare pentru obiectele software este cea de ***clasă***, fiecare obiect fiind o instanță a clasei din care face parte. O clasă reprezintă un șablon după care sunt create obiectele care sunt instanțe ale acelei clase; pentru fiecare clasă vor fi specificate proprietățile și metodele obiectelor respective.

Obiecte în JavaScript

Limbajul *JavaScript* consideră limbajul HTML ca fiind orientat obiect. Să considerăm următorul exemplu:

```
<HTML>
  <HEAD>
    <TITLE>
      Titlul paginii
    </TITLE>
  </HEAD>
  <BODY>
    <A href="http://www.domeniu.com">Domeniu
    </A>
    <FORM>
      <INPUT type="button" name="Buton">
    </FORM>
  </BODY>
<HTML>
```

Din punctul de vedere al limbajului *JavaScript*, în documentul HTML anterior sunt definite următoarele obiecte:

- `document.title`
- `document.form`
- `document.form.buton`

Obiectul navigator

În *JavaScript* există un obiect care descrie caracteristicile programului de navigare și anume `navigator`. Câteva dintre proprietățile acestui obiect sunt `appName`, `appVersion`, `appCodeName` și `userAgent`. Alături de

aceste proprietăți există câteva proprietăți suplimentare care depind de programul de navigare folosit. În cazul programului *Internet Explorer 5.0* alte proprietăți ale obiectului `navigator` sunt: `appMinorVersion`, `onLine`, `opsProfile`, `cpuClass`, `systemLanguage`, `plugins`, `cookieEnabled`, `platform`, `userProfile`, `mimeTypes` și `userLanguage`. În cazul programului *Netscape Navigator 4.5* proprietățile suplimentare sunt: `language`, `platform`, `securityPolicy`, `plugins` și `mimeTypes`. Următorul document HTML ilustrează modul în care pot fi aflate proprietățile obiectului `navigator`, precum și valorile lor:

```
<HTML>
  <HEAD>
    <TITLE>
      Obiectul Navigator
    </TITLE>
  </HEAD>
  <BODY>
    <SCRIPT language="JavaScript">
      for (propertyName in navigator) {
        document.write(propertyName+" =
          "+navigator[propertyName]+"<BR>");
      }
    </SCRIPT>
  </BODY>
</HTML>
```

Urmărind figurile 6 și 7 puteți vedea diferențele dintre paginile Web generate de *Internet Explorer 5.0*, respectiv *Netscape Navigator 4.5*.

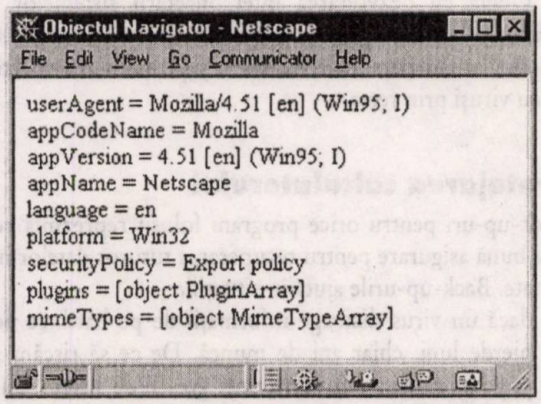


fig. 6 Proprietățile obiectului `navigator` în *Netscape Navigator 4.5*

Obiectul location

Un alt obiect predefinit în *JavaScript* este `location`, obiect care poate fi folosit pentru identificarea documentului curent. Cele mai folosite proprietăți ale acestui obiect sunt `protocol`, `hostname`, `port`, `pathname`, `search` și `hash`; în funcție de programul de navigare folosit, pot exista proprietăți suplimentare.

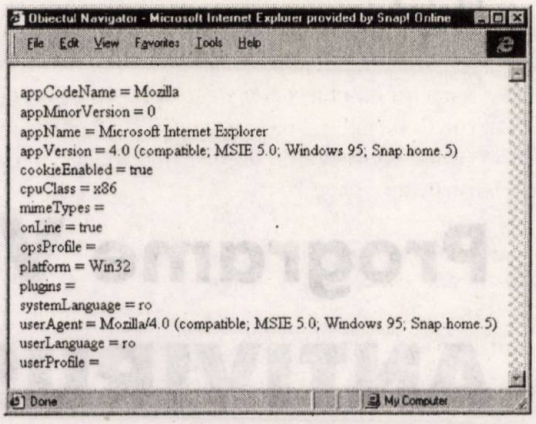


fig. 7 Proprietățile obiectului `navigator` în *Internet Explorer 5.0*

Obiecte corespunzătoare câmpurilor din formulare

De obicei, câmpurile unui formular sunt descrise în interiorul unui marcaj `<FORM>`; după cum am mai spus elementele unei pagini Web sunt tratate de către limbajul *JavaScript* ca fiind obiecte, deci există posibilitatea accesării și chiar a modificării proprietăților lor.

Unul dintre cel mai des întâlnite câmpuri ale formularelor este căsuța de validare; din punct de vedere al limbajului *JavaScript* există posibilitatea selectării sau deselectării casetei de validare, precum și posibilitatea verificării stării casetei (selectată sau deselectată). Pentru a selecta sau deselecta o casetă de validare, instrucțiunea corespunzătoare are forma:

`document.formular.buton.value="etichetă".`

Unul dintre cele mai interesante tipuri de elemente ale unui formular HTML îl reprezintă câmpurile ascunse; acestea pot fi folosite pentru a transmite anumite informații fără ca vizitatorul paginii să-și dea seama că aceste informații sunt transmise. Deși aceste câmpuri erau folosite pentru a face transferuri neautorizate de informații, noile facilități de securitate ale programelor de navigare descurajează asemenea practici; totuși, uneori, prezența unor astfel de câmpuri este absolut necesară. Să presupunem că există un script CGI care prelucrează datele dintr-un formular și trimite rezultatele la o anumită adresă e-mail; vizitatorul nu are nevoie să știe care este această adresă, în schimb această informație trebuie să fie disponibilă pentru script-ul CGI respectiv. Evident, ar fi fost posibil ca această informație să fie păstrată pe server, dar, în acest caz, script-ul ar fi putut fi folosit numai pentru a trimite datele la această adresă; în cazul în care script-ul primește adresa împreună cu restul datelor din formular, el poate fi folosit pentru a trimite date la orice adresă. Pentru a include un câmp ascuns în cadrul unui formular HTML, veți folosi o secvență de genul:

```
<INPUT type="hidden" name="mail" value="adr">
```

Mihai Scorțaru este student la Universitatea Tehnică Cluj. Poate fi contactat prin e-mail la adresa smihai@asterix.obs.utcluj.ro.

Programe VIRUS și ANTIVIRUS

Angela Pop

Se spunea că virușii devastatori de baze de date, produse program și alte informații prețioase au fost create de persoane care sperau ca în schimbul "devirusării" să obțină recompense substanțiale. Numai că antiviruși pot produce și alții...

Virusul

Virusul este un program nedorit și auto-instalabil, care se răspândește de la un calculator la altul, atașându-se unui alt program sau unei componente a sistemului. În acest fel, virusul supraviețuiește și se dezvoltă. Un virus de calculator are următoarele caracteristici:

1. auto-clonarea - se poate reproduce;
2. de obicei are nevoie de un program-gazdă și supraviețuiește prin intermediul acestuia;
3. distruge controlul și funcțiile programului gazdă, în interesul virusului.

Virusul infestează sistemul unui calculator și prin executarea unei aplicații infectate, virusul se poate răspândi în alte sisteme. Un program virusat poate contamina prin intermediul dischetelor, al casetelor, al modemului sau prin rețea. La rularea programelor infestate, virusul încearcă să se reproducă.

Virușii calculatoarelor, spre deosebire de cei biologici, au fost creați de oameni spre a distruge. Virușii sunt niște programe create ca și exercițiu de anumite persoane care vor să compromită programe soft.

Acțiunea virușilor

Nu se poate stabili un prototip pentru viruși. Orice program de acest calibru este în felul lui unic. Există totuși asemănări în ceea ce privește acțiunea virușilor:

1. infestarea gazdei;
2. supraviețuirea și propagarea;
3. transmiterea infecției și la alte aplicații;
4. posibile efecte distrugătoare.

Istoria virușilor

Ideea de auto-clonare a apărut la sfârșitul anilor 1940, când John von Neuman a creat primul program cu capacitate de auto-reproducere, stimulând astfel considerabil interesul în această problemă. La început, acest program a fost un exercițiu academic, apoi s-a dezvoltat într-un joc numit "core wars" în Laboratoarele Bell. Erau două programe oponente, fiecare având alocată o parte din memoria calculatorului. Fiecare program avea două obiective: să supraviețuiască și să-l distrugă pe celălalt. O mare parte din strategia de supraviețuire o constituia auto-clonarea. În anii '60-'70 s-au emis multe studii despre auto-reproducerea virușilor și despre folosirea lor în mod nociv.

Odată cu dezvoltarea interconectării sistemelor, a crescut și interesul pentru viruși. Cu creșterea numărului de calculatoare personale, s-a mărit și posibilitatea infectării cu viruși prin rețea.

Protejarea calculatorului

Back-up-uri pentru orice program folosit reprezintă cea mai bună asigurare pentru recuperarea tuturor datelor infectate. Back-up-urile ajută în situațiile:

1. dacă un virus distruge informații de pe hard, se pot pierde luni, chiar ani de muncă. De ce să riscăm să pierdem munca din cauza unei greșeli de sistem sau a unui virus?
2. back-up-urile ne ajută să readucem sistemele la un stadiu precedent.

O mai bună abordare ar fi protejarea sau blocarea dischetelor; orice aplicație pe care o rulați trebuie să fie *write-protected*.

Programele de protecție împotriva virușilor sunt, de asemenea, foarte utile. O nouă piață pentru astfel de aplicații a fost dată de nevoia de programe anti-virale. Unul dintre primele tipuri de produse este cel care anunță userul de prezența virusului. Chiar dacă e eficient în analiză,

acest soft nu este recomandat pentru utilizare curentă, deoarece tinde să genereze alarme false.

Următorul tip de programe anti-virus este scannerul, care examinează fiecare byte din orice aplicație. Scannerul are o bază de date cu semnele particulare ale virușilor. Dacă găsește astfel de particularități la un anume program, avertizează user-ul și câteodată dă și opțiuni de distrugere a virusului. Scanarea soft-ului ușurează mult munca individuală, dar și acest produs are multe contraindicații. Dacă apare un virus nou despre care scannerul nu știe nimic, nu îl detectează.

Un alt tip de soft anti-virus urmărește schimbările din fișierele sistem, detectând și noii viruși. Cele mai bune exemple ar fi: *Data Physician*, de *Digital Dispatch Inc.* și *CodeSafe*, de *Chris Ware*.

Instalarea de soft posibil infectat

Când instalați o aplicație nouă, nu riscați! Fie că luați soft de pe net, fie din alte surse, examinați programul înainte de a-l încărca.

Se recomandă citirea cu atenție a acordului de licență, multe programe implică riscuri prea mari, pentru a le trece cu vederea.

De altfel, ar fi recomandabil ca orice program soft nou să fie încercat timp de 30 de zile pe un sistem izolat și abia apoi instalat pe PC-ul operațional.

Principalele condiții ce trebuie îndeplinite de programul anti-virus

Pentru a fi cât mai funcțional un program anti-virus trebuie să îndeplinească următoarele condiții:

1. să ofere securitate;
2. să solicite efort minim din partea utilizatorului;
3. să poată fi funcțional, atât pentru produsele *driver based* aflate pe server, cât și pentru cele aflate pe stații
4. să se instaleze/dezinstaleze rapid;
5. să nu ocupe mult spațiu pe hard-disk și să utilizeze puțină memorie;
6. să aibă opțiune de remediere a fișierelor afectate;
7. să asigure deplina funcționalitate a opțiunii "*help*";
8. să nu implice riscuri din partea utilizatorului.

Impactul programelor virus și importanța existenței anti-virușilor

În ordine cronologică, în lumea soft-ului, în primul rând și-au făcut apariția virușii sau programele care declanșează erori. Prezența acestora a revoluționat întreaga societate a calculatoarelor. Ca o replică, programele anti-vir și-au făcut apariția, aducând, odată cu trecerea timpului, din ce în ce mai multe utilități. Datorită invaziei mari de viruși, prezența anti-virușilor apare ca o achiziție vitală și ca o adevărată modalitate de salvare a titanilor precum Microsoft, mereu vânați de viruși compromițători.

Angela Pop este elevă în clasa a X-a la Liceul "Traian Lalescu" din Reșița. Poate fi contactată prin e-mail la adresa: angelapop@mail.rtns.ro

NEWS...

Windows 2000

Release Candidate 2

În data de 15 septembrie a fost lansată o nouă versiune beta a sistemului de operare *Windows 2000* produs de compania *Microsoft*. Compania recunoaște că această nouă versiune beta a *W2K* nu este pregătită pentru implementări scalabile. Totuși Microsoft afirmă că *Windows 2000* se află "pe ultima sută de metri" și promite că noul sistem de operare va fi lansat la sfârșitul acestui an.

Microsoft a lansat Vizact 2000

o nouă aplicație pentru editarea paginilor Web

Compania *Microsoft* a anunțat marți, 7 septembrie 1999, disponibilitatea unei noi aplicații desktop care permite utilizatorilor să adauge documentelor MS *Office* caracteristici multimedia și pentru Internet, fără a mai fi nevoie de a scrie cod *HTML*. *Vizact 2000*, care este evaluat la un preț de 149$, poate facilita publicarea pe Web a fișierelor *Word*, *Excel*, *Outlook*, sau altele, care pot să includă imagini, animație sau elemente interactive cu utilizatorul. Spre deosebire de *FrontPage*, care este versiunea profesională a acestui pachet, *Vizact* a fost proiectat pentru a permite și celor neinițiați să adauge caracteristici de acest tip în propriile documente. Compania *Great American Office Supply*, care a testat noul produs, a declarat că este atât de simplu de folosit încât te simți "idiot". Pentru a putea fi folosit, *Vizact 2000* are nevoie doar de *Microsoft Internet Explorer 5.0*.

Vă place GInfo?

Nume: _____ Prenume: _____ Vârsta: _____

Adresa: _____

Localitate: _____ Cod poştal: _____ Judeţ: _____ Telefon: _____

Ocupaţia: elev informatică ☐ elev ☐ student ☐ cadru didactic ☐ altă ocupaţie ☐ (precizaţi: _____)

Şcoala (firma, organizaţia etc.): _____

Dotări

Folosiţi un calculator acasă? Da ☐ Nu ☐

Câte calculatoare sunt instalate în şcoală: (aprox.) _____

Aveţi acces la e-mail: Da ☐ Nu ☐

Aveţi acces la WWW: Da ☐ Nu ☐

Cum apreciaţi dotarea cu tehnică de calcul a şcolii: _____

Aprecieri generale

Acordaţi note de la 1 la 10 pe următoarele criterii:

Structura tematică: ___

Calitatea articolelor: ___

Prezentare grafică: ___

Calitate tipografică: ___

Consideraţi că preţul este:

prea mare ☐ accesibil ☐ prea mic ☐

De cât timp cunoaşteţi Gazeta de Informatică (ani)? _____

Cum aţi procurat acest număr:

chioşc ☐ abonament ☐ cadou ☐ altfel ☐

În acest număr

Apreciaţi cu note de la 1 la 10 articolele acestui număr:

Memorii cache ___

I. Concursuri ___

II. Hardware & Software ___

Algoritmi de vacanţă ___

Principiul inducţiei în informatică ___

Principiul cutiei lui Dirichlet ___

Pagini WEB cu JavaScript ___

Programe virus şi antivirus ___

Articolul care v-a plăcut cel mai mult: _____

De ce? _____

Articolul care v-a plăcut cel mai puţin: _____

De ce? _____

Consideraţi că ponderea problemelor rezolvate este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că ponderea problemelor propuse este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că ponderea subiectelor teoretice este:

prea mică ☐ potrivită ☐ prea mare ☐

Consideraţi că nivelul de accesibilitate al articolelor este

prea cobotât ☐ potrivit ☐ prea ridicat ☐

În viitor

În următoarele numere aşteptaţi:

mai multe articole de programare? Da ☐ Nu ☐

mai multe articole teoretice? Da ☐ Nu ☐

mai multe articole despre Internet? Da ☐ Nu ☐

mai multe prezentări de programe? Da ☐ Nu ☐

prezentări de jocuri? Da ☐ Nu ☐

articole despre hardware? Da ☐ Nu ☐

V-ar plăcea să citiţi:

pagini redactate de elevii unor licee? Da ☐ Nu ☐

lucrări premiate, realizate de elevi? Da ☐ Nu ☐

actualităţi şcolare? Da ☐ Nu ☐

prezentarea unor situri Web? Da ☐ Nu ☐

tips & tricks? Da ☐ Nu ☐

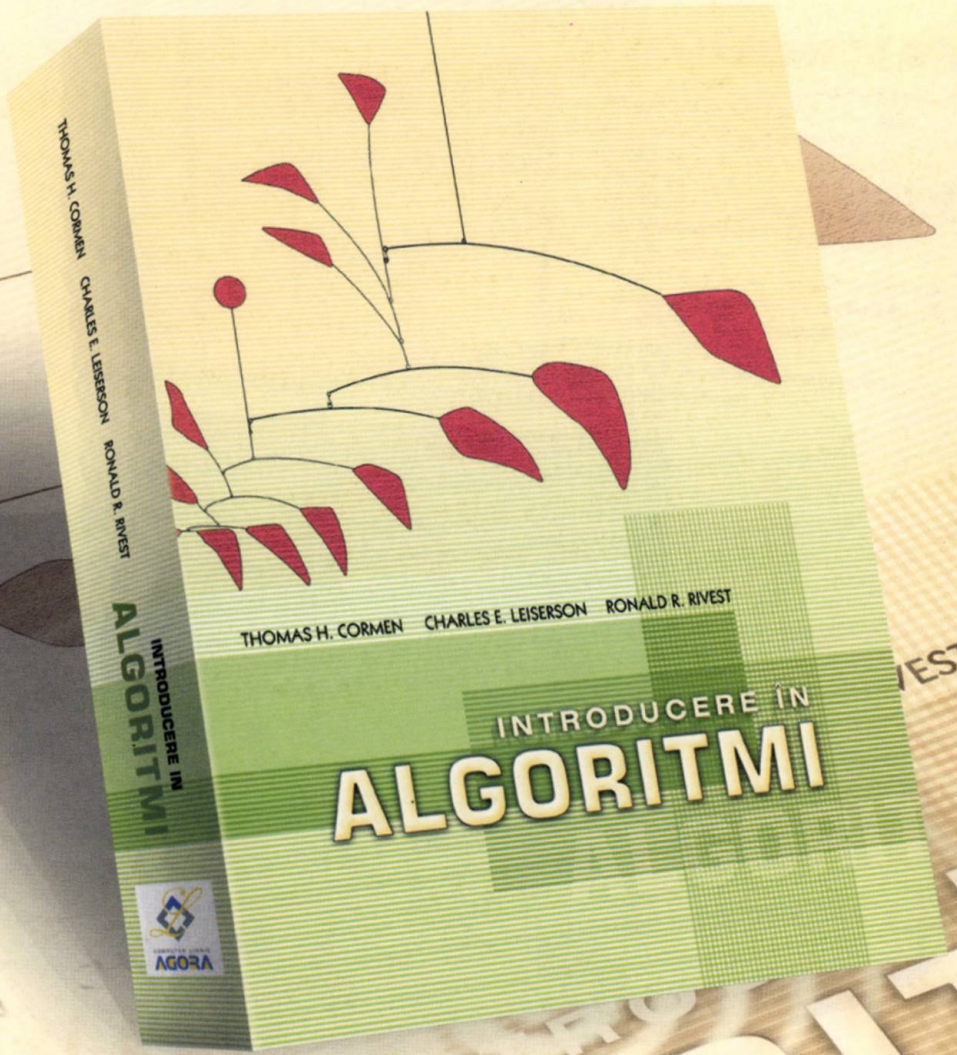
Comentarii:

Dacă nu vă este suficient spaţiul, folosiţi coli albe suplimentare

COMPUTER LIBRIS AGORA

Este în curs de apariție și în limba română celebra carte de ALGORITMI scrisă de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, publicată de MIT Press, SUA. Este o carte care de la prima ei apariție a fost retipărită, numai în Statele Unite, de 21 ori.

Această carte oferă o introducere exhaustivă a studiului modern al algoritmilor. Analiza și proiectarea algoritmilor tratați este accesibilă și asigură condițiile unui studiu, pe de o parte fundamental, pe de altă parte profund.

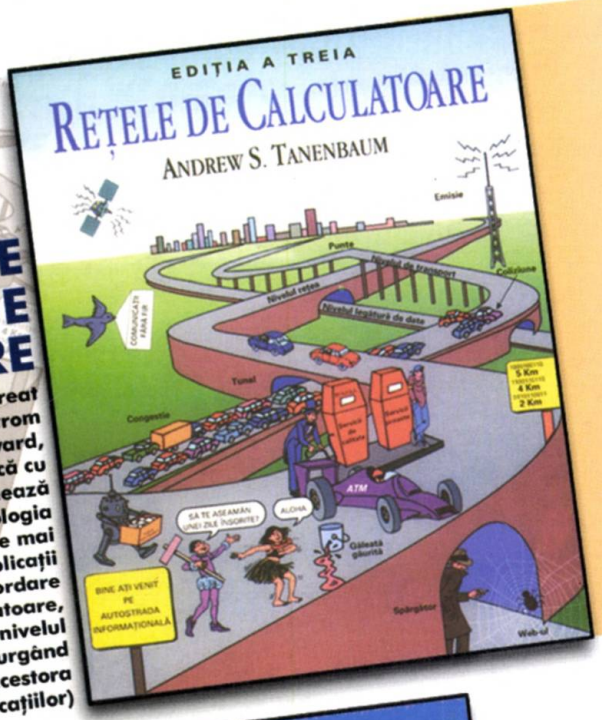


COMPUTER LIBRIS AGORA

Pentru comenzi sau informații suplimentare puteți suna la tel. 064-192422

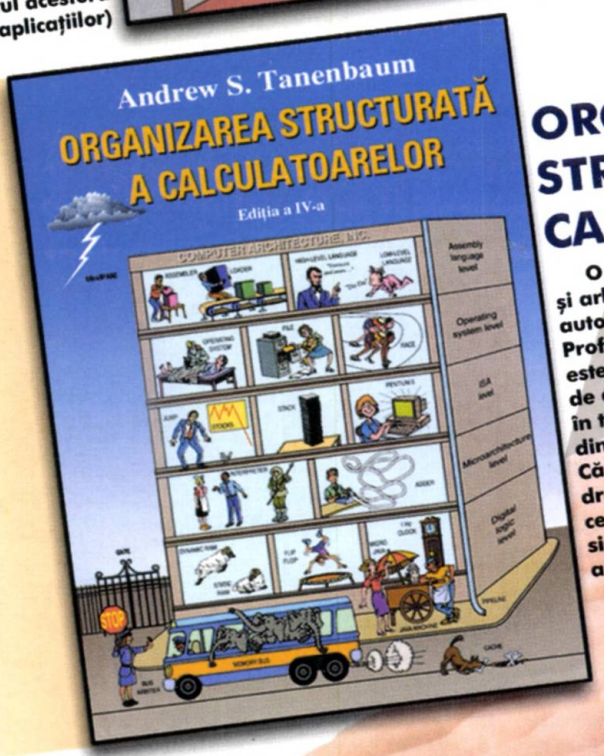
COMPUTER PRESS AGORA

ANDREW S. TANENBAUM



REȚELE DE CALCULATOARE

Autor, profesor și cercetător, laureat al premiului ACM Karl V.Karlstrom Outstanding Educator Award, Andrew S.Tanenbaum explică cu grijă felul în care rețelele funcționează în interior, pornind de la tehnologia hardware până la cele mai utilizate aplicații de rețea. Cartea urmează o abordare structurată a rețelelor de calculatoare, începând de jos, (de la nivelul fizic) și parcurgând treptat calea spre vârful acestora (nivelul aplicațiilor)



ORGANIZAREA STRUCTURATĂ A CALCULATOARELOR

O carte celebră de inițiere în structura și arhitectura calculatoarelor scrisă de un autor nu mai puțin celebru. Profesorul Andrew Tanenbaum este unul din cei mai faimoși autori de cărți de specialitate cunoscut în toate facultățile de calculatoare din lume.
Cărțile sale sunt utilizate drept cărți de referință pentru cel puțin trei domenii: rețele, sisteme de operare și arhitectura sistemelor de calcul.

COMPUTER PRESS AGORA
Str. T.Vladimirescu Nr.63/1 (CP 230/1) • Tel. 065-166516 Fax 065-166290
www.agora.ro

Noiembrie 1999
Vol. 9/7

Lei 15.000

gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:

NUMERE PRIME
o problemă încă nerezolvată

ALGORITMI
la nivel de bit

CONCURSURI

TEHNO
Grafica Directcolor în Pascal

ȘTIRI

SERIAL
JavaScript

Bursele Agora!
Concurs pentru elevi organizat de Gazeta de Informatică
Runda 2

COMPUTER LIBRIS AGORA

MANUAL Clasa a IX -a
CONFORM PROGRAMEI ȘCOLARE

Informatică

PROFIL MATEMATICĂ-INFORMATICĂ

DOINA RANCEA
GABRIELA BÂLAN
IOANA CUCU
DIANA GRUIȚĂ

AGORA

Manualul este astfel conceput încât să permită învățarea noțiunilor prezentate cu ușurință, realizarea unor aplicații utile, individual sau în grupuri de elevi. Sunt incluse multe probleme rezolvate și propuse din care profesorul poate să aleagă, în funcție de nivelul general al clasei și importanța și dificultatea problemelor. Problemele mai dificile de la sfârșitul fiecărui capitol ar putea fi date elevilor ca „miniproiecte" de aplicații în scopul deprinderii lor cu munca în echipă. Autorii acestui manual au căutat să ofere un instrument didactic prin utilizarea căruia să crească eficiența proceselor de predare și învățare și să împartă propria lor experiență didactică, acumulată pe parcursul a peste 25 de ani.

Limbaj
Algo
fundar

Clasa a X -a

MANUAL Clasa a X -a
CONFORM PROGRAMEI ȘCOLARE

Limbajul Pascal
Algoritmi fundamentali

LIMBAJE DE PROGRAMARE ȘI BAZELE INFORMATICII
APLICAȚII PRACTICE DE LABORATOR

DOINA RANCEA

AGORA

Manualul prezintă tehnici și metode de programare ca: recursivitate, alocare dinamică, metodele backtracking, programare dinamică, greedy, divide et impera și branch and bound. Prezentarea este accesibilă, iar aplicațiile sunt diverse: grafică (fractali), jocuri, teoria grafurilor. Manualul a fost conceput, plecând de la ediția 1993 (Limbajul Turbo Pascal, clasa a X-a) care a fost regândit substanțial și îmbogățit, astfel încât să fie accesibil unui elev de clasa a X-a. Ediția anterioară s-a bucurat de un succes deosebit, fiind apreciată de mii de cititori.

Două noi apariții

Pentru comenzi și informații suplimentare ne puteți contacta la tel. 064-192422

COMPUTER LIBRIS AGORA

Vol. 9, Nr. 7/1999

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Au mai colaborat:
Mihai Oltean (Cugir)
Florin Sabău (Reșița)
Valentin Gheorghiță (Ploiești)

Tehnoredactare:
Dan Crețu, Călin Borz
Coperta și grafica:
Mircea Drăgoi

Adresa pentru corespondență:
C.P. 230-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: (065) 166.516
Fax: (065) 166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *Master Druck*
Târgu-Mureș
tel: 065/162.359
Editor: Computer Press Agora srl
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Gabriela Bucșa (gbucsa@agora.ro)
Gabriel Piscupescu (gpiscup@agora.ro)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: (01) 330.92.81
Fax: (01) 330.92.85

sumar

probleme.. PAG. 4

propuse pentru REZOLVARE

Baraje de selecție pentru Olimpiada Internațională de Informatică

concurs.. PAG. 28

RUNDA 2

Bursele AGORA!
Concurs pentru elevi și studenți, organizat de Gazeta de Informatică

soluții.. PAG. 10

probleme REZOLVATE

Soluțiile unor probleme propuse în cadrul taberei de pregătire de la Mediaș

tehno.. PAG. 30

Grafică Directcolor în Pascal
Florin SABĂU

Modurile Highcolor și Truecolor

focus.. PAG. 19

Algoritmi la nivel de bit
Mihai OLTEAN

Numere prime - o problemă încă nerezolvată
Valentin GHEORGHIȚĂ

serial.. PAG. 36

Pagini WEB cu JavaScript
Mihai SCORȚARU

Vectori, operații matematice și operații cu șiruri de caractere

actual.. PAG. 45

I. Concursuri

II. Software

III. Hardware

Din 25 octombrie 1999
Preț Super Special pentru abonați
449 $
Calculator Alpis (fără monitor) în configurația prezentată în campania de abonamente Agora
BRINEL

BRINEL Business Partners:
Cluj-Napoca: BRINEL, Blvd. N. Titulescu #4, tel: 064-414.610
* Alba-Iulia: Info DSC SRL, 058-827.144
* Baia Mare: Multinet SRL, 062-222.201
* București: BRINEL, 094-547.804
* Brașov: Cosidor Computer, 068-310.534
* Dej: Eustel Prod Com, 064-433.713
* Râmnicu Vâlcea: Cibpro SRL, 050-734.894
* Sibiu: Alfasoft SA, Sucursala, 069-213.962
* Suceava: Globe Seattle PC, 030-215.497
* Târgu Mureș: Electro-Orizont SRL, 065-168.807
* Timișoara: AS Computer

The most productive C++ tool for Windows 2000 and the Web.

Put power and productivity to work with Visual C++.

Microsoft® Visual C++® 6.0 is an award-winning tool that helps you develop the highest-performance business solutions for the Windows® 2000 operating system and the Web.

More Productive Development

- **Maximize productivity.** Speed development time and write less code with new IntelliSense® technology—get on-the-fly programming assistance in the editor.
- **Get faster turnaround.** Edit code right in your debugging session and continue without rebuilding, allowing you to maintain your application's state.

More Productive Development

- **Ensure your Web applications are scalable and reliable.** Develop scalable, Web-based applications with new transaction support for Microsoft Transaction Server (MTS) and Internet Information Server (IIS).
- **Customize your Internet solutions.** Take advantage of new classes and wizards for easy customization of your Microsoft Internet Information Server–based applications with MFC.

Create Faster Applications

- **Achieve peak performance.** New compiler optimizations have been added to ensure that executables run more quickly. And you can compile large projects up to 30 percent faster without sacrificing any optimizations.
- **Take advantage of flexible data access.** Use OLE DB wizards to get the highest-performance access to any data source.

WIN100 WINDOWS MAGAZINE

Win100 Award

The best and most innovated product of the year, May 1999

Game Developer front line AWARD 1998

FrontLine Award,

Game Developer Magazine, June, 1998

Datamation Product of the Year

Datamation Product of the Year Award (Visual C++ Enterprise Edition)

Datamation magazine, April, 1998.

What's new in Visual C++:

Windows 2000 Developer's Readiness Ki

Learn how to take advantage of new Windows 2000 features through in-depth, step-by-step training and technical resources —find out how to gain greater scalability, lower cost deployment and maintenance, a the confidence that your current applications are Windows 2000 ready.

- *Get a jump-start on the new Active Directory Service Interfaces (ADSI).*
- *Learn how to build distributed applications with COM+, including integrated component management services support for transaction processing between objects, a support for dynamic load balancing of objects across servers.*
- *Take advantage of powerful distributed application services such as message queuing, object pooling, an more. Learn to create self-repairing applications with new Visual Studio Installer that features easy installati rollback and repair.*
- *Get an overview of the new Windows 2000 Web Services, Core Services, Microsoft Management Console, and much more.*

Microsoft Data Engine (MSDE)

Create desktop and shared solutions that easily scale to SQL Server 7.0 using the new Microsoft Data Engine (MSDE):

- *100% compatible with SQL Server 7.0.*
- *Royalty-free distribution of your MSDE-based applications.*
- *Reliability and features of SQL Server 7.0.*

MSDN Library Subscription

Get the essential programming information you need at yo fingertips for Windows 2000 and beyond with the MSDN™ Library.

- Now includes a year's worth of updates deliver to you via an MSDN Library Subscription

Visit the Microsoft® Visual C++® Web site at:
http://msdn.microsoft.com/visualc/

1999 Microsoft Corporation. All rights reserved. Microsoft, ActiveX, BackOffice, MSDN, Visual Basic, Visual FoxPro, Visual InterDev, Visual J++, Visual SourceSafe, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

Microsoft

Manualul și CALCULATORUL

Manualul

A sosit și primul an școlar în care se aplică (pentru clasa a IX-a) noua programă școlară, conform planului de învățământ aprobat anul trecut. În noul sistem de învățământ preuniversitar unele clase ale liceelor teoretice au profilul *Matematică-Informatică* și printre cunoscutele discipline de cultură generală au o disciplină denumită *Informatică*. Această disciplină se studiază în doar trei ore pe săptămână, număr de ore care pare insuficient pentru cunoștințele pe care profesorii de informatică ar dori să le predea elevilor din clasele a IX-a. Și nu doar profesorii sunt "de vină". La rândul lor, mulți elevi solicită cunoștințe noi, diversificate din cele mai variate domenii ale informaticii. Calculatorul, prin minunata lume a fiabilității, a răbdării infinite, a jocurilor și a Internetului, îi provoacă permanent pe cei tineri și îi atrag spre aplicații complexe. Alții - ambițioși și doritori să cucerească premii și medalii - vor să participe la concursuri, deci solicită cunoștințe temeinice de algoritmică și programare.

Dar nu despre acest aspect mi-am propus să scriu, ci despre faptul că Ministerul Educației Naționale a organizat prima acțiune care s-a finalizat cu *aprobarea* unor manuale școlare pentru mai multe discipline, printre care și pentru disciplina *Informatică*.

Această aprobare a fost acordată de *Comisia de Avizare a Manualelor*, comisie formată din profesori universitari și de liceu. Pentru manuale de informatică au cerut avizare mai multe edituri, printre care amintim: *Teora, All, Editura Didactică și Pedagogică, L&S* și *Computer LIBRIS Agora*. Toate manualele au primit acordul editării cu însemnul "avizat" și este și firesc să fie așa. Majoritatea autorilor sunt profesori cu o bogată experiență didactică și care au gândit aceste manuale nu doar ca o colecție de cunoștințe ci ca pe un instrument util în predarea, dar mai ales în învățarea conținuturilor prevăzute în programa școlară. Aceste manuale se pot procura din librării sau de la edituri, fiecare profesor având libertatea să aleagă și să recomande elevilor săi acea carte pe care o consideră cea mai potrivită, compatibilă cu propriul stil de predare.

Calculatorul

Pentru a învăța *matematică* este nevoie de manuale și de culegeri. În cazul *informaticii* mai avem o "pretenție": calculatoare. Și nu orice fel de calculatoare! De exemplu, *VisualBasic* nu se poate învăța pe un 386... Dar nu se poate programa nici în *Delphi*, nici într-un alt mediu vizual. "Nici nu trebuie" gândesc mulți dintre cititorii acestor rânduri... Dar cum altfel reușește învățământul să pregătească tineretul pentru așteptările anilor de după 2000? Cine poate să spere că aici se oprește dezvoltarea informaticii? Cine poate să creadă că în viitor va trebui în continuare ca toți programatorii să *redescopere* toți algoritmii? Dimpotrivă, sper că vin vremuri "bune", când programatorii va trebui să știe înainte de toate "ce anume știe" un anumit mediu, ce anume este "ascuns" în bibliotecile software-ului folosit și să asambleze verigă după verigă ceea ce a fost realizat de alții, nemaifiind nevoie să se "reprogrameze" totul. Va crește eficiența muncii informaticianului care va avea o productivitate spectaculoasă. Nu cred că, de exemplu pentru *metoda backtracking* este valabilă sintagma "fără tabla înmulțirii nu se poate..." Ba da, cred că se poate și fără backtracking, și fără teoria grafurilor. În "schimbul lor" trebuie aduse în școli calculatoare puternice pe care să ruleze aplicații multimedia atractive, care să dovedească tinerilor că într-adevăr, calculatoarele ne ușurează existența. Deocamdată, noi profesorii, și "calculatoarele" de astăzi din școli, reușim doar să "le facem capu' mare"...

Deci ne trebuie echipament, începând cu școala primară, și terminând cu liceul. De unde? Veți întreba... *De la noi toți.* Cum adică? Nu ați auzit de proiectul lansat de *Procter&Gamble*, Ministerul Educației Naționale, organizația *Salvați copiii!*, IBM, PRO TV, alte firme și organizații? Cumpărați... și copiii din școlile noastre vor avea calculatoare.

Dar nu este suficient să "ajutăm" holdingul amintit să realizeze încasări cât mai mari... Este nevoie de proiecte interesante, înaintate la Ministerul Educației Naționale, din care să rezulte ce anume se va realiza pe și cu aceste calculatoare în școala respectivă... Da, înainte de toate, dacă tot cerem ceva, trebuie să știm pentru ce cerem. Apoi trebuie să avem idei despre modalitățile de realizare ale planurilor de viitor. Și mai trebuie să fim în stare să dovedim că suntem capabili să realizăm ce am visat...

Clara Ionescu

PROBLEME propuse pentru REZOLVARE

Vă prezentăm în continuare enunțurile celor nouă probleme pe care au încercat să le rezolve membrii Lotului Național de Informatică în cadrul barajelor de selecție pentru Olimpiada Internațională de Informatică.

P079901: Arbore

lect. univ. Clara Ionescu, Univ. "Babeș Bolyai", Cluj

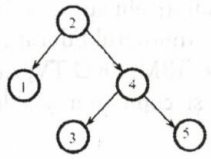
Se consideră un arbore binar de căutare A având n noduri conținând cheile 1, 2,..., n. O permutare $p = [p_1,\ldots,p_n]$ a numerelor întregi 1, 2,..., n se numește *consistentă* cu arborele A dacă arborele de căutare poate fi construit pornind de la arborele vid prin inserarea numerelor întregi p_1, $p_2,\ldots,p_n$ în această ordine.

Să se determine numărul permutărilor mulțimii {1, 2,..., n} care sunt consistente cu un arbore dat. (Altfel spus: câte secvențe distincte de chei există pentru ca arborii binari de căutare - obținuți prin inserarea cheilor în ordinea dată - să fie identici cu arborele dat?)

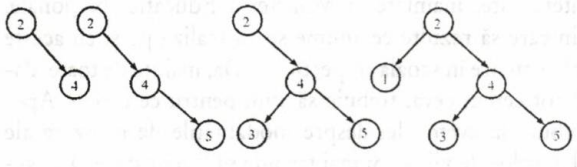
Exemplu

Arborele din figură are exact 8 permutări consistente.

De exemplu, permutările 2, 1, 4, 3, 5 și 2, 4, 5, 3, 1 sunt consistente cu arborele din figura:



Inserând elemente consecutive din permutarea 2, 4, 5, 3, 1, se obține arborele dat:



Date de intrare

Pe prima linie a fișierului **ARBORE.IN** este scris un singur număr natural n ($1 \le n \le 100$), reprezentând numărul de noduri din arbore. Pe cea de a doua linie se află n numere naturale distincte, separate printr-un singur spațiu. Aceste numere reprezintă cheile asociate arborelui dat în preordine. Primul număr este cheia asociată rădăcinii; este urmată de cheile asociate subarborelui stâng, și de cheile subarborelui drept (de asemenea în preordine).

Date de ieșire

Rezultatul se va scrie în fișierul **ARBORE.OUT** și trebuie să reprezinte numărul permutărilor consistente cu arborele din fișierul de intrare.

Atenție: Acest număr poate avea până la 200 de cifre.

Exemplu

ARBORE.IN	ARBORE.OUT
5 2 1 4 3 5	8

Explicație

Cele 8 permutări consistente cu arborele dat sunt:

1) 2 1 4 3 5;
2) 2 1 4 5 3;
3) 2 4 1 3 5;
4) 2 4 3 1 5;
5) 2 4 5 3 1;
6) 2 1 4 5 3;
7) 2 4 1 5 3;
8) 2 4 3 5 1.

P079902: La cinema

Cristian Cadar, student Univ. Politehnică București

Un grup de n prieteni se duc la cinema. Fiecare dintre ei și-a cumpărat bilet separat, însă conform înțelegerii inițiale toți și-au luat bilete în rândul 17. Ajungând după începutul filmului, ei se așează la întâmplare pe acest rând. Fiecare dintre cei n prieteni consideră că locul de pe biletul său este cel mai bun, așa că fiecare dorește să ajungă pe locul său. Pentru a nu deranja prea tare restul spectatorilor ei se hotărăsc ca în fiecare minut, mai multe perechi de persoa

ne să-și schimbe locurile între ele. La fiecare minut, numărul de schimbări de locuri poate fi oricât de mare, însă o persoană nu poate participa decât la o singură astfel de schimbare.

Aflați numărul minim de minute necesare pentru ca fiecare persoană să ajungă pe locul său.

Pentru simplificare, considerăm că persoana 1 are biletul cu locul 1, persoana 2 are biletul cu locul 2, și așa mai departe, persoana n are biletul cu locul n.

Date de intrare

Datele de intrare se citesc din fișierul **CINEMA.IN**. Pe prima linie este scris numărul n (n≤1000) de prieteni care merg la cinema. Pe fiecare din următoarele n linii se află câte un număr, reprezentând locul pe care prietenii s-au așezat inițial.

Date de ieșire

Datele de ieșire se scriu în fișierul **CINEMA.OUT**. Prima linie a acestui fișier va conține numărul minim M de minute necesare pentru ca fiecare persoană să ajungă pe locul său. Urmează M blocuri cu următoarea structură:

- prima linie a blocului conține numărul R de mutări care sunt executate la minutul respectiv;
- următoarele R linii conțin perechi de numere (i j), separate printr-un spațiu, având semnificația: în minutul respectiv, persoana i își schimbă locul cu persoana j.

Exemplu

CINEMA.IN	CINEMA.OUT
3	2
3	1
1	1 2
2	1
3	2 3

P079903: D'Artagnan

prof. Dana Lica, Liceul "I. L. Caragiale", Ploiești

Când D'Artagnan avea 18 ani s-a hotărât să se ducă la Paris și să devină mușchetar. Singura modalitate de a călători era de a folosi șoselele regale care legau orașele. Fiecare șosea care lega două orașe avea (la ambele capete ale sale) porți care permiteau accesul în orașele respective. Fiecare astfel de poartă era păzită fie de mușchetarii regelui, fie de gardienii cardinalului. Rolul acestora era să noteze numele persoanelor care intrau în oraș. Din această cauză, dacă cineva intra într-un oraș pe o poartă păzită de mușchetari, era obligat să părăsească orașul pe o poartă păzită tot de mușchetari. Dacă nu ar fi respectat regula, la ieșirea pe o poartă păzită de gardieni ar fi fost arestat, deoarece nu figura pe lista persoanelor care au primit, din partea lor, permisiunea de intrare în oraș; dacă o persoană intra pe o poartă păzită de gardieni, era obligată să părăsească orașul tot pe o poartă păzită de gardieni.

Pentru ca situația lui D'Artagnan să fie și mai complicată, trebuie știut faptul că între mușchetari și gardieni existau mari neînțelegeri. Din acest motiv orice șosea dintre două orașe avea la un capăt o poartă păzită de gardieni, iar la celălalt capăt o poartă păzită de mușchetari.

D'Artagnan a stat ore întregi în fața hărții pentru a căuta cel mai scurt drum pentru a ajunge la Paris. Evident, el putea părăsi orașul în care locuia, prin orice poartă.

În Franța existau n orașe numerotate cu numere de la 1 la n., Parisul numerotat cu 1, iar orașul lui D'Artagnan cu n. Existau m șosele care legau câte două orașe și pe care se putea circula în ambele sensuri. Pentru fiecare șosea se cunosc: numerele de ordine ale orașelor pe care le unește, lungimea acesteia și tipul păzitorilor de la poarta orașului de plecare (evident la celălalt capăt se vor afla opozanții).

Scrieți un program care determină lungimea minimă a unui drum care respectă condițiile de deplasare descrise, precum și orașele pe care le străbate.

Date de intrare

Pe prima linie în fișierul text **DART.IN** este scris numărul n (1≤n≤50) de orașe. Pe a doua linie este scris numărul m (1≤m≤1000) de șosele. Pe următoarele m linii sunt descrise șosele în formatul i j L t, semnificând o șosea între orașele i și j de lungime L cu poarta păzită de t la ieșirea din orașul i. Datele sunt separate printr-un singur spațiu. i, j și L sunt numere naturale, iar t este un caracter: M pentru mușchetari, G pentru gardieni.

Date de ieșire

Pe prima linie a fișierului **DART.OUT** se va scrie lungimea minimă a drumului determinat. Pe cea de-a doua linie se vor scrie numerele de ordine ale orașelor în ordinea în care trebuie parcurse de D'Artagnan. Numerele vor fi despărțite printr-un singur spațiu.

În cazul în care problema admite mai multe drumuri minimale în fișier se va scrie un singur drum, iar în cazul în care problema nu admite soluții, fișierul de ieșire va conține mesajul: `Fara solutie`

Exemplu

DART.IN	DART.OUT
5	5
7	5 4 1
1 2 3 M	
1 4 4 G	
1 3 2 G	
2 5 3 G	
2 4 10 G	
5 4 1 G	
4 3 1 M	

P079904: Evadatul

lect. univ. Clara Ionescu, Univ. "Babeș Bolyai", Cluj

Un prizonier încearcă să evadeze din închisoare. El iese prin sistemul de canalizare format din M tuneluri și N inter-

secții de tuneluri. Fiecare tunel leagă două intersecții diferite și nu există două tuneluri diferite între aceeași pereche de intersecții.

Prizonierul se găsește inițial în intersecția numărul 1 și trebuie să ajungă la intersecția numărul N unde se găsește ieșirea.

Prizonierul are o bilă grea legată de picior printr-un lanț de lungime L. Pentru a reduce zgomotul, lanțul trebuie să fie în permanență întins. Bila se târăște pe măsură ce prizonierul înaintează și dacă lanțul trece prin (intră și iese) patru sau mai multe intersecții simultan frecarea lanțului este atât de mare încât prizonierul nu mai poate înainta. În plus, prizonierul nu poate face stânga împrejur în interiorul unui tunel; pe de altă parte în momentul în care el intră într-o intersecție, nu poate pleca de acolo prin tunelul prin care a intrat.

Inițial, bila se găsește în intersecția numărul 2. Există un tunel de lungime exact L care leagă intersecțiile 1 și 2, lanțul găsindu-se inițial întins de-a lungul acestui tunel.

Scrieți un program care stabilește dacă prizonierul poate evada (sau nu) din închisoare. În caz afirmativ, determinați lungimea celui mai scurt drum de evadare.

Date de intrare

Pe prima linie a fișierului **EVADAT.IN** se află trei numere naturale: N ($2<N\leq40$) reprezentând numărul intersecțiilor, M ($1\leq M\leq780$) reprezentând numărul tunelurilor, și L ($0<L\leq30000$) reprezentând lungimea lanțului.

Pe următoarele M linii sunt descrise tunelurile. Fiecare linie conține trei numere naturale: a, b ($0<a,b\leq N$) și d ($0<d\leq30000$) semnificând faptul că există un tunel de lungime d între intersecțiile a și b.

Date de ieșire

Fișierul **EVADAT.OUT** va conține un singur număr care reprezintă lungimea celui mai scurt drum de evadare (între intersecțiile 1 și N). Dacă nu există un asemenea drum, în fișier se va scrie numărul -1.

Exemplu

EVADAT.IN	EVADAT.OUT
8 8 10 1 2 10 1 3 2 3 4 2 4 7 2 4 5 6 5 6 4 6 4 4 7 8 2	22

P079905: Fire

Valentin Gheorghiță, student Univ. Politehnică, București

Un electrician folosește n ($2<n\leq500$) fire pentru cablarea casei din pod până în beci. Capetele firelor sunt numerotate de la 1 la n atât în pod cât și în beci. Deoarece electricianul este o persoană extrem de împrăștiată, un fir având capătul numerotat cu i în pod, nu are întotdeauna capătul din beci numerotat cu i.

Electricianul poate urca în pod sau poate coborî în beci, inițial aflându-se în pod. Sarcina voastră este de a-l ajuta pe electrician să găsească corespondența capetelor din pod cu cele din beci, executând un număr minim de urcări și coborâri. În scopul găsirii capetelor corespunzătoare, electricianul (aflat în pod sau în beci) fie conectează două fire (dacă firul x este conectat cu firul y și firul y este conectat cu firul z, atunci firul x este conectat automat cu firul z), fie verifică dacă două fire sunt conectate la capătul opus.

Aveți la dispoziție un *unit* (**FIRE.PAS**) care va conține următoarele funcții:

Procedure Start
: procedură de inițializare; orice altă funcție sau procedură va fi apelată numai după apelul procedurii **Start**;

Function Number:Integer
: funcția care returnează numărul de fire;

Procedure Connect(x,y:Integer)
: procedura care permite conectarea firului x cu firul y;

Function IsConnect(x,y:Integer):Boolean;
: funcția care verifică dacă firele x și y sunt conectate la capătul opus;

Procedure Clear
: procedura care permite decuplarea tuturor legăturilor existente în locul în care se află electricianul (pod sau beci);

Procedure GoUp
: procedura care produce mutarea electricianului din beci în pod;

Procedure GoDown
: procedura care produce mutarea electricianului din pod în beci;

Procedure Solution(x,y:Integer)
: procedura prin care specificați că firul x aflat în pod corespunde firului y din beci; această procedură trebuie apelată pentru toate firele;

Procedure Stop
: procedura care semnalează finalul rezolvării; programul vostru nu va fi punctat decât după apelarea acestei proceduri.

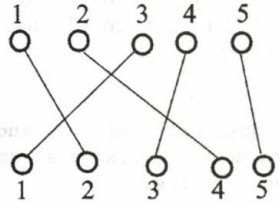
Listing: FIRE.PAS

```
unit fire;

Interface

Function Number:Integer;
Procedure Clear;
Procedure GoUp;
Procedure GoDown;
Procedure Connect(x,y:Integer);
Procedure Start;
Procedure Stop;
Function IsConnect(x,y:Integer):Boolean;
```

```
Procedure Solution(x,y:Integer);

Implementation

var sol:array[1..2,1..500] of Integer;
    elev:array[1..500] of Integer;
    leg_down,leg_up:array[1..500] of Integer;
    op,n,i:Integer;
    f:Text;
    inceput,up,down:Boolean;

procedure Raspuns(s:string;p:Integer);
begin
  Writeln(s); Writeln('Punctaj: ',p);
  Halt
end;

function Number:Integer;
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit!',0);
  Number:= n
end;

Procedure Clear;
var i:Integer;
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit!',0);
  if up then for i:=1 to n do leg_up[i]:=0;
  if down then for i:=1 to n do leg_down[i]:=0
end;

procedure GoUp;
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit!',0);
  if up then raspuns('Electricianul nu poate
              urca mai sus!',0);
  down:=false;
  op:=op+1; up:=true
end;

procedure GoDown;
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit!',0);
  if down then Raspuns('Electricianul nu poate
               cobora mai jos!',0);
  down:=true; op:=op+1; up:=false
end;

procedure Start;
begin
  Writeln('Conexiune realizata!');
  inceput:=true
end;
Procedure Connect(x,y:Integer);
var i,tmp:Integer;
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit!',0);
  if (x<1) or (x>n) or (y>n) or (y<1)
  then Raspuns('Se incearca conectarea a unor
       fire inexistente!',0);
  if x=y
  then Raspuns('Nu puteti conecta un fir cu
       el insusi!',0);
  if up
  then
    if leg_up[x]=0
    then
      if leg_up[y]=0
     then begin leg_up[x]:=x; leg_up[y]:=x end
      else leg_up[x]:=leg_up[y]
    else
      if leg_up[y]=0
      then leg_up[y]:=leg_up[x]
      else
      begin
        tmp:=leg_up[y];
        for i:=1 to n do
          if leg_up[i]=tmp
          then leg_up[i]:=leg_up[x]
      end;
  if down
  then
    if leg_down[x]=0
    then
      if leg_down[y]=0
      then
      begin leg_down[x]:=x; leg_down[y]:=x end
      else leg_down[x]:=leg_down[y]
    else
      if leg_down[y]=0
      then leg_down[y]:=leg_down[x]
      else
      begin
        tmp:=leg_down[y];
        for i:=1 to n do
          if leg_down[i]=tmp
          then leg_down[i]:=leg_down[x]
      end
end;

function IsConnect(x,y:Integer):Boolean;
var i,tmp:Integer;
    conect:Boolean;
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit!',0);
  if (x<1) or (x>n) or (y>n) or (y<1)
  then Raspuns('Testare pentru fire
       inexistente!',0);
  if x=y then Raspuns('Un fir nu poate fi
              conectat cu el insusi ! ',0);
  conect:=false;
  if up
  then
    if (leg_down[sol[1,x]]<>0) and
       (leg_down[sol[1,x]]=leg_down[sol[1,y]])
    then conect:=true;
  if down
  then
    if (leg_up[sol[2,x]]=leg_up[sol[2,y]]) and
       (leg_up[sol[2,x]] <>0)
    then conect:=true;
  IsConnect:=conect
```

```
end;
procedure Solution(x,y:Integer);
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit!',0);
  if (x<1) or (x>n) or (y>n) or (y<1)
  then Raspuns('Solutie pentru fire
       inexistente!',0);
  elev[x]:=y
end;

procedure Stop;
var corect:Boolean;
    i:Integer;
begin
  if not inceput
  then Raspuns('Informatii cerute fara a fi
       realizata conexiunea la unit ! ',0);
  inceput:=false; corect:=true;
  Assign(f,'ELECTR.OUT'); Rewrite(f);
  Writeln(f,'Numarul de fire:',n);
  Write(f,'Solutie corecta  ');
  Writeln(Solutie concurent');
  for i:=1 to n do
  Writeln(f,'  ',sol[1,i]:3,'     ',elev[i]:3);
  Close(f);
  for i:=1 to n do
    if elev[i]<>sol[1,i] then corect:=false;
  if not corect
  then Raspuns('Solutia furnizata este
       incorecta!',0);
  Writeln('Numarul de drumuri este ',op);
  if op>2
  then Raspuns('Numarul de drumuri este
       incorect!',0);
  Raspuns('Solutie corecta!',10)
end;

Begin
  op:=0; up:=true; down:=false; inceput:=false;
  Assign(f,'ELECTR.IN'); Reset(f); Readln(f,n);
  for i:=1 to n do leg_up[i]:=0;
  for i:=1 to n do leg_down[i]:=0;
  for i:=1 to n do Read(f,sol[1,i]);
  Readln(f);
  for i:=1 to n do Read(f,sol[2,i]);
  Close(f)
End.
```

Exemplul pe care vi-l oferă unit-ul anterior este cel din figură:



P079906: Ora de gimnastică

Cristian Cadar, student Univ. Politehnică București

După ora de germană urmează ora de gimnastică!!! Cei n elevi prezenți la oră sunt așezați în cerc, unii stau în picioare, alții sunt așezați în genunchi. Ei poartă tricouri numerotate de la 1 la n. Exercițiul presupune realizarea de genoflexiuni în felul următor: într-o aceeași unitate de timp oricare elev își va schimba poziția (sau nu) în funcție de poziția sa, de poziția celui aflat cu două poziții în stânga sa și în funcție de poziția colegului aflat imediat în dreapta sa. Schimbările corespunzătoare unei unități de timp au loc simultan.

Copiii se identifică prin numărul lor de ordine în cerc, număr desenat și pe tricoul lor. (În dreapta copilului având numărul i pe tricou, urmează cel cu numărul i+1, excepție făcând copilul cu numărul de ordine n, în dreapta căruia se află copilul cu numărul de ordine 1.)

Scrieți un program care determină poziția fiecărui elev după un număr cunoscut k de unități de timp.

Date de intrare

Pe prima linie a fișierului **GYM.IN** se află numărul k ($0 \le k \le 2.000.000.000$) reprezentând numărul unităților de timp.

Pe cea de-a doua linie se află o secvență de atâtea caractere câți copii participă la exercițiu ($5 \le n \le 10000$). Al i-lea caracter indică poziția celui de-al i-lea elev ('J' pentru jos și 'S' pentru sus).

Pe fiecare din următoarele linii se află câte patru caractere cu următoarea semnificație: dacă un anumit elev se află în starea indicată de primul caracter de pe linie, colegul aflat cu două poziții în stânga sa se află în starea indicată de cel de-al doilea caracter de pe linie și colegul din dreapta sa se află în starea indicată de cel de-al treilea caracter de pe linie, atunci elevul respectiv își va schimba poziția în starea indicată de cel de-al patrulea caracter de pe linie.

Date de ieșire

Fișierul de ieșire **GYM.OUT** conține n caractere semnificând starea în care se află cei n copii după k unități de timp.

Exemplu

GYM.IN	GYM.OUT
2	SJJSJ
JSJJS	
JJJJ	
JJSJ	
JSJJ	
JSSS	
SJJJ	
SJSS	
SSJS	
SSSJ	

P079907: Origami

Radu Lupșa, preparator Univ. "Babeș Bolyai", Cluj

Origami este o veche artă japoneză ce constă în îndoirea unei simple foi de hârtie dreptunghiulare și obținerea unor forme de animale, flori, sau alte figuri. În zilele noastre, un amator de origami și de calculatoare a construit o mașină capabilă să fabrice origami, executând în acest scop o

secvență de instrucțiuni simple.

Mașina constă dintr-o masă pe care se plasează inițial o coală pătrată de hârtie și un dispozitiv de îndoire. Colțul stânga jos al hârtiei are coordonatele `(0,0)`, iar colțul dreapta sus are coordonatele `(100,100)`. Mașina primește o secvență de instrucțiuni; fiecare instrucțiune descrie câte o îndoire. Fiecare instrucțiune constă din coordonatele a două puncte, p_1 și p_2. Îndoirea se face în lungul liniei ce unește p_1 și p_2, partea din dreapta liniei (considerate dinspre p_1 către p_2) îndoindu-se peste partea din stânga.

Una din problemele amatorului nostru de origami este aceea de a găsi în final un loc unde să perforeze origami-ul rezultat în scopul agățării lui într-un cui. În locul ales, grosimea origami-ului, adică numărul de pliuri suprapuse, trebuie să fie nici prea mică - caz în care origami-ul riscă să se rupă sub greutatea proprie -, nici prea mare - caz în care perforația este dificil de executat. Amatorul nostru alege deci o secvență de puncte și vrea să afle care este grosimea origami-ului în fiecare dintre aceste puncte. Scrieți un program care, plecând de la instrucțiunile de confecționare și de la coordonatele punctelor alese, calculează grosimea origami-ului în punctele date.

Date de intrare

Datele se vor citi din fișierul **ORIGAMI.IN**. Prima linie a acestui fișier conține numărul `n` (`0≤n≤8`) de îndoiri. Fiecare din următoarele `n` linii conține coordonatele x_1, y_1, x_2, y_2 ale punctelor p_1, respectiv p_2. Pe următoarea linie se găsește numărul `m` (`1≤m≤50`) de puncte de test (candidate pentru puncte de perforație). Pe fiecare din următoarele `m` linii se găsesc coordonatele punctelor de test.

Notă: coordonatele punctelor sunt numere reale; distanța dintre un punct de test și marginea hârtiei, respectiv față de o îndoitură nu este mai mică de `0.01`.

Date de ieșire

Ieșirea se va face în fișierul **ORIGAMI.OUT**. Pentru fiecare punct de test se va scrie o linie pe care se va afișa grosimea origami-ului în acel punct.

Exemplu

ORIGAMI.IN	ORIGAMI.OUT
2	4
-0.5 -0.5 1 1	2
1 75 0 75	0
6	2
10 60.1	0
80 60	2
40 30	
10 20	
50 40	
20 40	

P079908: Permanent

Mihai Scorțaru, student Univ. Tehnică Cluj

Permanentul unei matrice este definit într-o manieră asemănătoare definiției determinantului; singura diferență constă în faptul că termenii care apar cu semnul - în dezvoltarea determinantului, vor apărea cu semnul + în dezvoltarea permanentului. În concluzie, formula pentru calculul permanentului unei matrice este:

$$perm(A) = \sum_{\sigma \in S} \prod_{i=1}^{n} a_{i\sigma_i}$$

unde `S` este mulțimea permutărilor mulțimii {1, 2,..., n}.

Se consideră matricea pătratică `A` de dimensiune `n` definită prin a_{ij}=`i*j`. Să se determine numărul de cifre (în baza 10) al permanentului acestei matrice.

Date de intrare

Fișierul de intrare **PERM.IN** conține numărul natural `n` (`1≤n≤24.572.457`).

Date de ieșire

Fișierul de ieșire **PERM.OUT** va conține, pe o linie, numărul de cifre al permanentului matricei `A`.

Exemplu

PERM.IN	PERM.OUT
2	1

Explicație

Pentru `n=2` matricea `A` este:

$$\begin{pmatrix} 1\cdot1 & 1\cdot2 \\ 2\cdot1 & 2\cdot2 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix},$$

iar permanentul ei este `perm(A)=1*4+2*2=8`, deci numărul de cifre este `1`.

P079909: Pitagora

Mihai Scorțaru, student Univ. Tehnică Cluj

Se consideră un număr natural `a` din intervalul [`1,32457`]. Să se găsească toate perechile de numere naturale `b` și `c` având proprietatea `a<b<c` pentru care există un triunghi dreptunghic care are laturi de lungimi `a`, `b`, `c`.

Date de intrare

Fișierul de intrare **PITAGORA.IN** conține numărul natural `a` (`1≤a≤32457`).

Date de ieșire

Fișierul de ieșire **PITAGORA.OUT** va conține, pe câte o linie, perechile de numere naturale `b` și `c` separate printr-un singur spațiu. Există întotdeauna cel puțin o astfel de pereche.

Exemple

PITAGORA.IN	PITAGORA.OUT
3	4 5
PITAGORA.IN	**PITAGORA.OUT**
9	12 15
	40 41

SOLUȚIILE problemelor propuse

Vă prezentăm în continuare câteva dintre soluțiile oficiale pentru problemele propuse în cadrul taberei de pregătire de la Mediaș. Soluțiile aparțin autorilor problemelor.

P069901: A. P. M.

Adăugarea unei muchii într-un arbore generează un ciclu. Presupunem că introducem o muchie între nodul `i` și nodul `j`.

Pentru ca APM-ul grafului astfel obținut să nu se schimbe, trebuie ca muchia de cost maxim de pe drumul de la `i` la `j` din arborele inițial să aibă costul mai mic sau egal cu muchia introdusă. Pe această observație se bazează întreaga strategie. Se determină muchiile de cost maxim de pe fiecare drum dintre oricare două puncte ale arborelui inițial; se ordonează crescător atât vectorul acestora cât și costurile muchiilor care ne sunt puse la dispoziție. Fișierul de ieșire va conține toate perechile de două noduri distincte pentru care avem la dispoziție o valoare mai mare sau egală cu muchia de cost maxim de pe drumul dintre cele două noduri.

Listing: APM.PAS

```
{$M 65000,0,655360}
program graf_apm;
const fin='APM.IN';
      fout='APM.OUT';
      max=100;
type muchie=record
              i,j:Byte;
              c:Integer
            end;
     sir=array[1..max*max div 2] of Integer;
var a:array[1..3*max] of muchie;
    sel:array[1..max] of Boolean;
    inceput,sfarsit:array[0..max] of Integer;
    b:array[1..max*max div 2] of muchie;
    v:^sir;
    n,m,pz:Integer;

procedure citire;
var f:Text;
    i:Integer;
begin
  Assign(f,fin); Reset(f);
  Readln(f,n);
  for i:=1 to n-1 do
  begin
    Readln(f,a[i].i,a[i].j,a[i].c);
    a[i+n-1].i:=a[i].j;
    a[i+n-1].j:=a[i].i;
    a[i+n-1].c:=a[i].c;
    a[i+2*n-2]:=a[i]
  end;
  m:=0;
  New(v);
  while not Seekeof(f) do
  begin
    Inc(m);
    Read(f,v^[m])
  end;
  Close(f)
end;

procedure sort1(l,r:Integer);
var i,j,x:Byte;
    y:muchie;
begin
  i:=l; j:=r;
  x:=a[(l+r) div 2].i;
  repeat
    while a[i].i<x do Inc(i);
    while x<a[j].i do Dec(j);
    if i<=j
    then
    begin
      y:=a[i];
      a[i]:=a[j]; a[j]:= y;
      Inc(i); Dec(j)
    end
  until i>j;
  if l<j then sort1(l,j);
  if i<r then sort1(i,r)
end;
procedure recurs(nivel,st,op:Byte;
```

```
                                    val:Integer);
var i:Integer;
begin
  if (st<op) and (nivel>2)
  then
  begin
    Inc(pz); b[pz].i:=st;
    b[pz].j:=op; b[pz].c:=val
  end;
  for i:=inceput[op] to sfarsit[op] do
    if not(sel[a[i].j])
    then
    begin
      sel[a[i].j]:=true;
      if val<a[i].c
      then recurs(nivel+1,st,a[i].j,a[i].c)
      else recurs(nivel+1,st,a[i].j,val);
      sel[a[i].j]:=false
    end
end;

procedure calcul_muchii;
var i,j:Integer;
begin
  j:=0;
  for i:=1 to 2*n-2 do
    if a[i].i<>j
    then
    begin
      inceput[a[i].i]:=i;
      sfarsit[j]:=i-1; j:=a[i].i
    end;
  sfarsit[j]:=2*n-2;
  pz:=0;
  Fillchar(sel,Sizeof(sel),false);
  for i:=1 to n do
  begin
    sel[i]:=true; recurs(1,i,i,0);
    sel[i]:=false
  end
end;

procedure sort2(l,r:Integer);
var i,j,x:Integer;
    y:muchie;
begin
  i:=l; j:=r;
  x:=b[(l+r) div 2].c;
  repeat
    while b[i].c<x do Inc(i);
    while x<b[j].c do Dec(j);
    if i<=j
    then
    begin
      y:=b[i]; b[i]:=b[j]; b[j]:=y;
      Inc(i); Dec(j)
    end
  until i>j;
  if l<j then sort2(l,j);
  if i<r then sort2(i,r)
end;
```

```
procedure sort3(l,r:Integer);
var i,j,x,y:Integer;
begin
  i:=l; j:=r;
  x:=v^[(l+r) div 2];
  repeat
    while v^[i]<x do Inc(i);
    while x<v^[j] do Dec(j);
    if i<=j
    then
    begin
      y:=v^[i]; v^[i]:=v^[j];
      v^[j]:=y; Inc(i); Dec(j)
    end
  until i>j;
  if l<j then sort3(l,j);
  if i<r then sort3(i,r)
end;

procedure solutie;
var f:Text; k,i:Integer;
begin
  Assign(f,fout); Rewrite(f);
  k:=m;
  for i:=pz downto 1 do
    if b[i].c<=v^[k]
    then
    begin
      Writeln(f,b[i].i,' ',b[i].j,' ',v^[k]);
      Dec(k);
      if k<1 then begin Close(f); Halt end
    end;
  Close(f)
end;

Begin
  citire;
  sort1(1,2*n-2);
  calcul_muchii;
  sort2(1,pz);
  sort3(1,m);
  solutie
End.
```

P069903: Spre culmi

Ideea de rezolvare este următoarea: se baleiază vectorul de la stânga la dreapta, pentru a se construi subșirurile cu o singură parcurgere. Fiecare element este adăugat la sfârșitul unuia dintre subșirurile deja formate sau, dacă acest lucru nu este posibil, el este pus separat pentru a începe un nou subșir.

Fie exemplul dat: N=7, V=(9, 2, 4, 7, 10, 11, 8). Atunci primul element (9) va fi în mod evident începutul unui subșir. De asemenea și 2, deoarece el nu poate fi pus în același subșir cu 9. Elementul 4 se leagă de 2 (este așezat în același subșir cu el), pentru că nu se poate lega de 9, și ar fi o pierdere inutilă să îl așezăm într-un subșir separat. Elementul 7 se leagă de subșirul deja format (2, 4). Ce se întâmplă însă cu elementul 10? El poate fi atașat atât subșirului (9), cât și subșirului (2, 4, 7).

În situațiile în care un element poate fi alipit la mai multe subșiruri, se preferă alipirea la subșirul care se termină într-un număr **cât mai mare**. De ce? Dacă alipim 10 la subșirul (2, 4, 7), obținem subșirurile (2, 4, 7, 10) și (9). Dacă însă alipim 10 la subșirul (9), obținem subșirurile (2, 4, 7) și (9, 10). În primul caz, capetele subșirurilor aveau valorile 10 și 9; în acest caz, capetele subșirurilor au valorile 7 și 10; această variantă este mai folositoare deoarece capetele subșirurilor sunt mai mici, ceea ce este bine pentru alipirea în continuare a altor numere (cum este numărul 8, de exemplu). Dacă s-ar fi alipit 10 la subșirul (2, 4, 7), s-ar fi obținut în final 3 subșiruri: (9), (2, 4, 7, 10, 11) și (8), adică o soluție neoptimă.

În concluzie, pentru fiecare element V[i] trebuie să aflăm, dintre toate subșirurile deja create, capătul cu valoarea cea mai mare, dar strict mai mică decât V[i]. Pentru aceasta se recomandă menținerea valorilor de la capetele subșirurilor într-un vector separat, sortat, asupra căruia se pot efectua căutări binare. De asemenea, vom reține pentru fiecare element din V succesorul său în subșirul crescător din care face parte.

Pentru fiecare element din vector se efectuează deci o căutare binară, ca urmare complexitatea algoritmului are ordinul $O(N \log N)$.

Listing: CULMI.PAS

```
program Crescatoare;
var V,Max,Next:array[0..10000] of Word;
    N,NSubs:Integer;

procedure ReadData;
var i:Integer;
begin
  Assign(Input,'CRESC.IN'); Reset(Input);
  ReadLn(N);
  for i:=1 to N do Read(V[i]);
  Close(Input)
end;

function BinSearch(W:Word;Lo,Hi:Integer):
                                    Integer;
{ Cauta in vectorul Max, care este          }
{ sortat descrescator, si returneaza        }
{ indicele celui mai mare element mai mic   }
{ decat W. Daca toate elementele sunt       }
{ mai mari ca W, atunci returneaza NSubs+1  }
var Mid:Integer;
begin
  while Lo<Hi do
  begin
    Mid:=(Lo+Hi) div 2;
    if W>V[Max[Mid]] then Hi:=Mid
                     else Lo:=Mid+1
  end;
  if (Lo>Hi) or (W<=V[Max[Hi]])
  then BinSearch:=Hi+1
  else BinSearch:=Hi
end;
```

```
procedure BuildVectors;
        { Construieste vectorul Next cu     }
        { inlantuirea subsirurilor cautate  }
var NewPlace,i:Integer;
begin
  NSubs:=0;
  Max[0]:=50001;
  for i:=1 to N do
  begin
    NewPlace:=BinSearch(V[i], 1, NSubs);
    if NewPlace>NSubs
    then Inc(NSubs)
    else Next[Max[NewPlace]]:=i;
    Max[NewPlace]:=i;
    Next[i]:=0
  end
end;

procedure WriteSolution;
 { Extrage subsirurile, folosind      }
 { informatia din vectorul Next.      }
 { Vectorul Max serveste aici numai ca }
 { vector caracteristic al elementelor }
 { deja tiparite                      }
var Marker,Ind,i:Integer;
begin
  for i:=1 to N do Max[i]:=0;
  Assign(Output,'CRESC.OUT');
  Rewrite(Output);
  WriteLn(NSubs);
  Marker:=0;
  for i:=1 to NSubs do
  begin
    repeat
      Inc(Marker)
    until Max[Marker]=0;
    Ind:=Marker;
    repeat
      Max[Ind]:=1;
      Write(Ind,' ');
      Ind:=Next[Ind]
    until Ind=0;
    Writeln
    end;
  Close(Output)
end;

Begin
  ReadData;
  BuildVectors;
  WriteSolution
End.
```

P069904: Apărați cetatea

Numărul minim de oșteni care trebuie plasați este egal cu valoarea fluxului maxim în multigraful orientat dat de problemă (în care sursa este orașul Brașov, iar destinația este orașul Mediaș). Demonstrația acestui fapt este foarte ușor de intuit dacă se cunoaște algoritmul Ford Fulkerson de aflare a fluxului maxim într-un graf.

Demonstrația se realizează în două etape:

Etapa 1

Vom arăta că sunt suficienți FM (FM reprezintă valoarea fluxului maxim) oșteni pentru a bloca accesul de la sursă la destinație.

Etapa 2

Vom arăta că nu putem bloca accesul de la sursă la destinație cu mai puțin de FM oșteni.

Etapa 2 este foarte simplu de arătat. Să presupunem prin reducere la absurd că putem bloca accesul de la sursă la destinație cu un număr de doar S oșteni (S<FM). Vom elimina arcele selectate (cele pe care vom amplasa oșteni) din graful inițial. Cu toate acestea, datorită modului de operare a algoritmului Ford Fulkerson, vom reuși să mai găsim drumuri de la sursă la destinație, cu o valoare totală de cel puțin FM-S. Astfel am găsit o contradicție cu ipoteza că cei S oșteni pot bloca accesul de la sursă la destinație.

Etapa 1 o vom demonstra prin selectarea efectivă a unui număr de arce având suma costurilor FM și care blochează accesul de la sursă la destinație. Pentru aceasta vom considera mulțimea A a nodurilor la care s-a ajuns la ultima iterație a algoritmului Ford Fulkerson. Fie X mulțimea vârfurilor grafului inițial și V mulțimea arcelor grafului inițial. Atunci mulțimea arcelor (i,j) cu proprietatea că (i,j) aparține lui V, i aparține lui S și j aparține lui X\S este o mulțime de arce cu proprietățile cerute. Demonstrația acestui fapt este de asemenea evidentă și se bazează pe modul de operare al algoritmului Ford Fulkerson.

În afara celor spuse mai sus, mai existau probleme referitoare la memorarea grafului, în așa fel încât algoritmul Ford Fulkerson să fie cât mai rapid. Clasica matrice de adiacență devenea, în cazul dimensiunii mari a datelor de intrare, imposibil de utilizat. O posibilă soluție era implementarea folosind listele de vecini. Pentru fiecare nod, era necesară folosirea a două liste de vecini (una pentru muchiile reale și una pentru muchiile de întoarcere) cu legături între ele, pentru a putea actualiza rapid drumurile găsite la fiecare iterație a algoritmului Ford Fulkerson.

Listing: CETATEA.PAS

```
{$M 65000,0,655360}
const NMax=5000;
type nod1=^tip_nod1;
     tip_nod1=record
                v,c,f:Word;
                urm:nod1
              end;
     nod2=^tip_nod2;
     tip_nod2=record
                v:Word;
                inv:nod1;
                urm:nod2
              end;
var i,j,k,n,c:Word;
    cap,cmin:Longint;
    ic1:array[1..NMax] of nod1;
    ic2:array[1..NMax] of nod2;
    f:Text;
    p:nod1;
    q:nod2;
    v:array[1..NMax] of Boolean;

procedure citire;
begin
  Assign(f,'MEDIAS.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do
  begin
    ic1[i]:=nil; ic2[i]:=nil
  end;
  while not Seekeof(f) do
  begin
    Readln(f,i,j,k);
    New(p);
    p^.v:=j; p^.c:=k; p^.f:=0;
    if ic1[i]=nil
    then
    begin ic1[i]:=p; ic1[i]^.urm:=nil end
   else begin p^.urm:=ic1[i]; ic1[i]:=p end;
    New(q);
    q^.v:=i; q^.inv:=p;
    if ic2[j]=nil
   then begin ic2[j]:=q; ic2[j]^.urm:=nil end
    else begin q^.urm:=ic2[j]; ic2[j]:=q end
  end;
  Close(f)
end;

procedure gaseste_si_prelucreaza_drum;
type coada=^tip_coada;
     tip_coada=record
                 v:Word;
                 leg:nod1;
                 direct:Boolean;
                 urm,ant:coada
               end;
var ic,sc,dd,start:coada;
    min:Word;

  procedure succesori;
  begin
    p:=ic1[ic^.v];
    while p<>nil do
    begin
      if not v[p^.v] and (p^.f<>p^.c)
      then
      begin
        v[p^.v]:=true;
        New(dd);
        dd^.v:=p^.v; dd^.leg:=p;
        dd^.direct:=true; dd^.urm:=nil;
        dd^.ant:=ic;
        sc^.urm:=dd; sc:=dd;
        if dd^.v=1 then Exit
      end;
      p:=p^.urm
```

```
    end;
    q:=ic2[ic^.v];
    while q<>nil do
    begin
      if not v[q^.v] and (q^.inv^.f<>0)
      then
      begin
        v[q^.v]:=true;
        New(dd);
        dd^.v:=q^.v; dd^.leg:=q^.inv;
        dd^.direct:=false;
        dd^.urm:=nil; dd^.ant:=ic;
        sc^.urm:=dd; sc:=dd;
        if dd^.v=1 then Exit
      end;
      q:=q^.urm
    end
  end;

begin          { gaseste_si_prelucreaza_drum }
  New(ic);
  ic^.v:=n;
  v[n]:=true;
  ic^.urm:=nil;
  start:=ic;
  sc:=ic;
  while (ic<>nil) and not v[1] do
  begin succesori; ic:=ic^.urm end;
  if not v[1] then Exit;
                { prelucreaza drum          }
                { gasesc minimul de pe drum }
  min:=65001;
  dd:=sc;
  while dd<>start do
  begin
    if dd^.direct
    then
      if min>dd^.leg^.c-dd^.leg^.f
      then min:=dd^.leg^.c-dd^.leg^.f
      else
    else
      if min>dd^.leg^.f
      then min:=dd^.leg^.f;
    dd:=dd^.ant
  end;
  cmin:=cmin+longint(min);
                  { scad minimul de pe drum }
  dd:=sc;
  while dd<>start do
  begin
    if dd^.direct
    then dd^.leg^.f:=dd^.leg^.f+min
    else dd^.leg^.f:=dd^.leg^.f-min;
    dd:=dd^.ant
  end;
  while start<>nil do
  begin
    dd:=start;
    start:=start^.urm;
    Dispose(dd)
  end
```

```
end;
procedure prelucrare;
begin
  cmin:=0;
  repeat
    Fillchar(v,Sizeof(v),false);
    gaseste_si_prelucreaza_drum
  until not v[1]
end;

procedure afisare;
begin
  Assign(f,'MEDIAS.OUT');
  Rewrite(f);
  Writeln(f,cmin);
  for i:=1 to n do
    if v[i]
    then
    begin
      p:=ic1[i];
      while p<>nil do
      begin
        if not v[p^.v]
        then Writeln(f,i,' ',p^.v,' ',p^.c);
        p:=p^.urm
      end
    end;
  Close(f)
end;

Begin
  Citire;
  Prelucrare;
  Afisare
End.
```

P069905: Axa OX

Soluția problemei este realizată prin metoda backtracking. Așezarea punctelor pe axă determină un arbore binar în care se va face căutarea soluției. Punctul de coordonate 0 și cel aflat la distanța maximă citită din fișierul de intrare, reprezintă puncte figurate pe axă.

Tehnica permite așezarea unui nou punct pe axă (simetric față de punctele extreme) dacă toate celelalte distanțe pe care le generează față de punctele deja figurate există. Vectorul a memorează distanțele (a[i] indică de câte ori apare distanța i). Odată cu așezarea unui punct pe axă se va face și decrementarea elementelor din a corespunzătoare.

Evident, dacă această acțiune nu este posibilă se încearcă poziția simetrică, sau se revine pe un nivel anterior.

Listing: AXA.PAS

```
{$M 65000,0,655360}
program axa;
const fin='AXA.IN'; fout='AXA.OUT';
type sir=array[0..20000] of Integer;
var a:array[1..20000] of Integer;
    i,n,maxim,nr:Longint;
    sol:array[0..20000] of Boolean;
    pus:^sir;
```

```
procedure citire;
var x:Integer;
    f:Text;
begin
  New(pus);
  Fillchar(a,Sizeof(a),0);
  Assign(f,fin); Reset(f);
  while not(Seekeof(f)) do
  begin
    Read(f,x);
    Inc(nr);
    Inc(a[x])
  end;
  n:=1;
  while n*(n-1)<>nr*2 do Inc(n)
end;

procedure tiparire;
var f:Text;
    i:Integer;
begin
  Assign(f,fout);
  Rewrite(f);
  for i:=0 to maxim do
    if sol[i]
    then Writeln(f,i);
  Close(f);
  Halt
end;

procedure cautare(max,pz:Integer);
var i,j:Integer;
    m:Boolean;
begin
  while a[max]=0 do Dec(max);
  if pz=n then tiparire;
  if not(sol[max])
  then
  begin
    m:=true;
    for i:=0 to pz-1 do
    if a[Abs(max-pus^[i])]>0
    then Dec(a[Abs(max-pus^[i])])
    else
    begin
      m:=false;
      Break
    end;
    if m and (i=pz-1)
    then
    begin
      pus^[pz]:=max;
      sol[max]:=true;
      cautare(max,pz+1);
      sol[max]:=false;
      Inc(i)
    end;
    for j:=0 to i-1 do
      Inc(a[Abs(max-pus^[j])])
  end;
  max:=maxim-max;
  if not(sol[max])
  then
  begin
    m:=true;
    for i:=0 to pz-1 do
      if a[Abs(max-pus^[i])]>0
      then Dec(a[Abs(max-pus^[i])])
      else
      begin
        m:=false;
        Break
      end;
    if (m and (i=pz-1))
    then
    begin
      pus^[pz]:=max;
      sol[max]:=true;
      cautare(maxim-max,pz+1);
      sol[max]:=false;
      Inc(i)
    end;
    for j:=0 to i-1 do
      Inc(a[Abs(max-pus^[j])])
  end
end;

Begin
  citire;
  for i:=1 to 20000 do
    if a[i]<>0
    then maxim:=i;
  Fillchar(sol,Sizeof(sol),false);
  sol[0]:=true;
  sol[maxim]:=true;
  Dec(a[maxim]);
  pus^[0]:=0;
  pus^[1]:=maxim;
  cautare(maxim,2)
End.
```

P069907: La grădiniță

I. Dacă noțiunea de lanț corect are accepțiunea 1)

Relația de recurență pe care se bazează determinarea numărului cerut:

`NR[i,j]:=NR[i-1,j]+NR[i,j-1];`

Avansând cu ea în tabloul NR, rezultatul îl obținem în componenta `NR[m,n]`

II. Dacă noțiunea de lanț corect are accepțiunea 2) relațiile de recurență căutate sunt mai dificil de obținut.

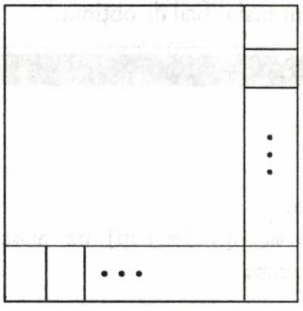
Listing: GRADINITA.PAS

```
program copii;
const max=50;
type numar=Comp;
     dimgrup=0..max;
var aa,a,b:array[dimgrup,dimgrup] of numar;
    i,j,k,m,n:dimgrup;
    c:numar;
    f:Text;
```

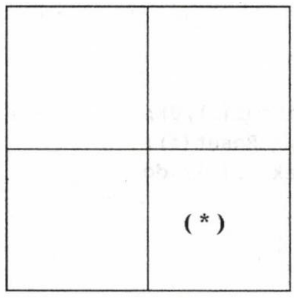
```
Begin
  Assign(f,'COPII.IN');
  Reset(f);
  Read(f,m,n);
  Close(f);
  Assign(f,'COPII.OUT');
  Rewrite(f);
                              { acceptiunea 1) }
  aa[1,1]:=1;
  for i:=2 to m do aa[i,1]:=i+1;
  for i:=2 to n do aa[1,i]:=i+1;
  for i:=2 to m do
    for j:=2 to n do
      aa[i,j]:=aa[i-1,j]+aa[i,j-1];
  Write(f,aa[m,n]:0:0,' ');
                              { acceptiunea 2) }
  a[1,0]:=0;
  a[1,1]:=1;
  b[1,1]:=1;
  for i:=2 to n do a[1,i]:=2;
  for i:=2 to m do
  begin
    a[i,0]:=1;
    b[i,1]:=2;
    a[i,1]:=a[i-1,1]+b[i-1,1]
  end;
  for i:=2 to n do
    b[1,i]:=a[1,i-1]+b[1,i-1];
  for i:=2 to m do
    for j:=2 to n do
    begin
      a[i,j]:=a[i-1,j]+b[i-1,j];
      b[i,j]:=a[i,j-1]+b[i,j-1]
    end;
  c:=a[m,n]+a[m,n-1];
  for k:=2 to n do
    c:=c+2*a[m,n-k];
  for k:=2 to m-1 do
    c:=c+b[m-k,n];
  Writeln(f,c:0:0);
  Close(f)
End.
```

P069908: Pătrate

Dacă n este număr par atunci partiția este următoarea:



Dacă n este impar atunci se face descompunerea:



iar pătratul (*) se descompune în n-3 pătrățele la fel ca în cazul anterior.

Listing: PATRAT.PAS

```
program patrat;
type tabl=array[1..100,1..100]of Byte;
var jum,dim,n,i,j,k:Integer;
    a:tabl;
    f:Text;
procedure afisare(a:tabl;dim:Integer);
var i,j:Integer;
begin
  Assign(f,'PATRAT.OUT');
  Rewrite(f);
  for i:=1 to dim do
  begin
    for j:=1 to dim do Write(f,a[i,j],' ');
    Writeln(f)
  end;
  Close(f)
end;

Begin
  Assign(f,'PATRAT.IN'); Reset(f);
  Readln(f,n);
  Close(f);
  for i:=1 to 100 do
    for j:=1 to 100 do a[i,j]:=1;
  if n mod 2=0
  then
  begin
    dim:=n div 2;
    k:=1;
    for i:=1 to dim do
    begin
      Inc(k);
      a[i,dim]:=k
    end;
    Dec(k);
    for i:=1 to dim do
    begin
      Inc(k); a[dim,i]:=k
    end;
    afisare(a,dim)
  end
```

```
  else
  begin
    dim:=n-3;
    jum:=dim div 2;
    for i:=1 to jum do
      for j:=jum+1 to dim do a[j,i]:=2;
    for i:=jum+1 to dim do
      for j:=1 to jum do a[j,i]:=3;
    for i:=jum+1 to dim-1 do
      for j:=jum+1 to dim-1 do a[i,j]:=4;
    k:=4;
    for i:=jum+1 to dim do
    begin Inc(k); a[i,dim]:=k end;
    for i:=jum+1 to dim-1 do
    begin Inc(k); a[dim,i]:=k end;
    afisare(a,dim)
  end
End.
```

P069910: La ora de germană

Soluția se bazează pe observația că dacă cuvintele au aceeași lungime atunci se poate aplica principiul optimalității. Problema se reduce la a determina într-un graf un drum prin K noduri, de cost maxim și care poate conține cicluri. Nodurile corespund cuvintelor, iar costul muchiei care leagă nodul I de nodul J este lungimea maximă a unui prefix din J care este sufix în I (cu alte cuvinte, cât poate "pătrunde" cuvântul I în cuvântul J).

Observație: Avem muchii și din I în I pentru orice I.

Aplicăm metoda programării dinamice, astfel: notăm cu A[I,J] costul maxim al unui drum care trece prin I noduri și ajunge în J. Avem condițiile inițiale A[0,J]=0 și relația de recurență:

A[I,J]=max{A[I-1,L]+D[L,J]|L=1,N},

iar maximul se regăsește într-una din celulele A[K,J], J=1,N. Lungimea minimă este K*L-MAX, iar cuvântul poate fi găsit ușor parcurgând o matrice "predecesor" P[I,J] pe care o construim în același timp cu matricea A.

Complexitatea algoritmului este $O(N^2*(K+T_i))$, unde T_i este timpul necesar determinării unei "intersecții" între două cuvinte. Acest lucru poate fi realizat trivial în $O(L^2)$, dar se poate realiza și în $O(L)$ folosind metoda de determinare a unui prefix-sufix maxim din algoritmul KMP pe cuvântul obținut prin alăturarea celor două cuvinte, respectiv pe un singur cuvânt în cazul unei bucle I→I.

Listing: GERMANA.CPP

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <alloc.h>
#define NMAX 100
#define LMAX 255
#define KMAX 1000
int i,j,t,k,n,l,mj,ls;
long *a,*aa,*tmp,min;
int b[NMAX+1][NMAX+1];
char *p[KMAX+1];
int d[KMAX+1];
char *c[NMAX+1];
char buf[BUFSIZ];
FILE *f;

void citeste(){
  f=fopen("GERMANA.IN","rt");
  fscanf(f,"%d %d %d",&n,&l,&k);
  for (i=1;i<=n;i++){
    c[i]= new char[l+1];
    if (!c[i]){
      puts("Out of memory");
      exit(1);
    }
    fscanf(f,"%s",c[i]);
  }
  fclose(f);
}

void build_b(){
  char s[LMAX*2+1];
  static int a[LMAX*2+10];
  int ci,cj;
  for (ci=1;ci<=n;ci++)
    for (cj=1;cj<=n;cj++){
      strcpy(s,c[ci]);
      if (ci!=cj)
        strcat(s,c[cj]);
      ls=strlen(s);
      a[0]=-1;
      a[1]=0;
      j=0;
      for (i=2;i<=ls;i++){
        while (j>=0 && s[j]!=s[i-1])
          j=a[j];
        j++;
        a[i]=j;
      }
      b[ci][cj]=a[ls];
    }
}

void rezolva(){
  build_b();
  aa=new long[n+1];
  for (i=1;i<=n;i++) aa[i]=1;
  a=new long[n+1];
  for (t=2;t<=k;t++){
    if (!(p[t]=(char*) farmalloc(n+1))) {
      puts("Out of memory!");
      exit(1);
```

```
    }
    for (i=1;i<=n;i++){
      min=10000000;
      for (j=1;j<=n;j++)
        if (min>aa[j]+1-b[i][j]){
          min=aa[j]+1-b[i][j];
          mj=j;
        }
      a[i]=min;
      p[t][i]=mj;
    }
    tmp=a;
    a=aa;
    aa=tmp;
  }
  min=10000000;
  for (j=1;j<=n;j++)
    if (min>aa[j]){
      min=aa[j]; mj=j;
    }
}

void scrie(){
  printf("%ld\n",farcoreleft());
  f=fopen("GERMANA.OUT","wt");
  fprintf(f,"%ld\n",min);
  for (i=k,j=mj;i>=1;i--){
    d[i]=j;
    if (i>1) j=p[i][j];
  }
  for (i=1;i<=k;i++)
    fprintf(f,"%s",c[d[i]]+b[d[i]][d[i-1]]);
  fclose(f);
}

void main(void){
  citeste();
  rezolva();
  scrie();
}
```

P069912: Numere

Problema generalizează modul de reprezentare a unui număr natural m într-o bază b.

Dacă se ia $x_1=x_2=\ldots x_n=b$ se obține:

$m=a_0+a_1b+a_2b_2+\ldots+a_kb_k$,

cu $0\le a_i<b$, pentru $0\le i\le k$.

Determinarea numerelor $a_0, a_1,\ldots, a_k$ se realizează folosind teorema împărțirii cu rest astfel:

$m=a_0+q_1x_1;$

$q_1=a_1+q_2x_2;$

...

$q_{k-1}=a_{k-1}+q_kx_k;$

$q_k=a_k+q_k+1x_k$

algoritmul se termină când $q_{k-1}=0$, sau k=n.

Dacă $q_{k-1}>0$, atunci problema nu are soluție. Condițiile $a_i<x_i+1$, sunt verificate datorită teoremei împărțirii cu rest.

Listing: NUMERE.PAS

```
program numere;
var f:Text;
    i,n,k,m,mm:Longint;
    ok:Boolean;
    x,a:array[0..1000] of Longint;

function corect:Boolean;
var i,baza,rez:Longint;
begin
  x[0]:=1;
  rez:=0;
  baza:=1;
  for i:=0 to k do
  begin
    baza:=baza*x[i];
    rez:=rez+a[i]*baza
  end;
  corect:=rez=mm
end;

Begin
  Assign(f,'NUMERE.IN'); Reset(f);
  Readln(f,m);
  mm:=m;
  Readln(f,n);
  for i:=1 to n do
    Read(f,x[i]);
  Close(f);
  Assign(f,'NUMERE.OUT');
  Rewrite(f);
  k:=0;
  ok:=true;
  while (k<n)and(m<>0) do
  begin
    a[k]:=m mod x[k+1];
    m:=m div x[k+1];
    k:=k+1
  end;
  k:=k-1;
  if (k<n)and(m=0) and corect
  then
  begin
    Writeln(f,'DA');
    Writeln(f,k);
    for i:=0 to k do
      Write(f,a[i],' ')
  end
  else Write(f,'NU');
  Close(f)
End.
```

Algoritmi la nivel de BIT

Mihai Oltean

Există două motive esențiale pentru care se recomandă folosirea algoritmilor la nivel de bit și anume: reducerea cantității de memorie folosite și creșterea vitezei de lucru.

Numere mici

Se dau două numere mici (cuprinse între 0 și 15). Se cere să se stocheze ambele numere, folosind un singur octet.

Soluție

Este evident că fiecare număr cuprins între 0 și 15 are nevoie de 4 biți pentru a fi reprezentat. De aceea un octet (care are opt biți) este suficient pentru a reprezenta două astfel de numere.

Vom stoca pe primii patru biți (cei mai puțini semnificativi) primul număr, iar pe următorii patru (cei mai semnificativi) următorul număr. Procedurile sunt următoarele:

```
function Codifica(a,b:Byte):Byte;
begin
  Codifica:=a+16*b
end;

procedure Decodifica(x:Byte; var a,b:Byte);
begin
  a:=x mod 16;
  b:=x div 16
end;
```

O altă metodă de codificare se bazează pe operațiile la nivel de bit, și anume, dacă în procedura următoare dorim să îl scriem pe a în primii 4 biți (cei mai puțin semnificativi), iar pe b în următorii 4 biți, atunci este nevoie să îl deplasăm pe b cu 4 biți spre stânga:

```
function Codifica(a,b:Byte):Byte;
begin
  Codifica:=a or b shl 4
end;
```

Decodificarea se face într-o manieră asemănătoare. Se știe că 15 = $00001111_{(2)}$, iar 240 = $11110000_{(2)}$

```
procedure Decodifica(x:Byte; var a,b:Byte);
begin
  a:=15 and x;
  b:=(240 and x) shr 4
end;
```

Un scurt program care utilizează procedurile de mai sus este:

```
var a,b,x:Byte; { declaratiile procedurilor }
Begin
  x:=Codifica(3,4);
  Writeln(x);
  Decodifica(x,a,b);
  Writeln(a,' ',b)
End.
```

Trasee comune

Doi prieteni A și B s-au hotărât să facă o excursie prinr-o țară cu N orașe. Fiecare și-a alcătuit singur intinerariul. Știind că un oraș este vizitat cel mult o dată de fiecare turist, se cere să se determine numărul legăturilor directe dintre orașele pe care ambii prieteni (A și B) le străbat.

Soluție

Pentru a simplifica explicația presupunem că pot fi vizitate cel mult 32 de orașe; acestea sunt numerotate de la 0 la 31. Traseele celor doi sunt date în vectorii drum1 și drum2 (sub forma unei înșiruiri de orașe). Noi trebuie să determinăm numărul legăturilor directe dintre două orașe care sunt conținute în ambele drumuri. Un algoritm care calculează acest lucru are o complexitate medie pătratică.

Putem reduce această complexitate folosindu-ne de o codificare mai specială a celor două drumuri. În loc să reținem un drum ca pe o succesiune de orașe, îl reținem ca pe o mulțime de legături directe dintre orașe. Aceste legături, la rândul lor, nu le reținem într-un șir, ci direct într-

o matrice a drumurilor, adică o matrice NxN unde pe poziția (i,j) avem valoarea 1 dacă între orașul i și orașul j există legătură directă și drumul turistului străbate această legătură. Deoarece avem două drumuri, vom construi două astfel de matrice. Dacă le reținem folosind efectiv două matrice de NxN octeți, atunci nu obținem nici o optimizare. Dar dacă pentru fiecare legătură folosim un singur bit, adică o linie a matricei drumurilor să fie, nu un șir de octeți, ci un șir de biți, atunci o primă optimizare s-a făcut în sensul reducerii memoriei folosite. Am spus că orașele sunt numerotate cu numere de la 0 la 31, deoarece în acest fel este suficientă o variabilă de tip Longint pentru a reprezenta o linie a matricei drumurilor. Dacă numărul orașelor este mai mare decât 32, atunci vor trebui mai multe variabile de tip Longint (sau Byte, Word etc.) pentru a reprezenta drumul.

A doua parte a optimizării constă în modul de a calcula perechile de legături directe care sunt conținute în cele două drumuri. Având la dispoziție cele două matrice de biți nu va trebui decât să aplicăm operația **and** între ele, rezultând astfel o nouă matrice care are biți 1 doar acolo unde ambele matrice inițiale au avut biți 1. Rezultatul se va obține numărând liniile care conțin o valoare diferită de 0.

Listing: DRUM.PAS

```
const MaxN=31;
type TDrum=array[1..MaxN+1] of Byte;
     TDrumBinar=array[0..MaxN] of Longint;
var drum1,drum2:TDrum;
    dBinar1,dBinar2,dBinar3:TDrumBinar;
    N:Byte;

procedure Codifica(d:TDrum;
                   var drumBinar:TDrumBinar);
var i:Byte;
begin
  for i:=1 to N-1 do
    drumBinar[d[i]]:=1 shl (31-d[i+1]);
  drumBinar[d[N]]:=1 shl (31-d[1])
end;

procedure Intersectie
          (drumBinar1,drumBinar2:TDrumBinar;
                   var drumBinar3:TDrumBinar);
var i:Byte;
begin
  for i:=0 to N-1 do
    drumBinar3[i]:=drumBinar1[i] and
                                drumBinar2[i]
end;

procedure Initializeaza(var d:TDrumBinar);
var i:Byte;
begin
  for i:=0 to N-1 do d[i]:=0
end;
function DetNrLegaturi
                (drumBinar:TDrumBinar):Byte;
var i,nrLeg:Byte;
begin
  nrLeg:=0;
  for i:=0 to N-1 do
    if drumBinar[i]<>0
    then Inc(nrLeg);
  DetNrLegaturi:=nrLeg
end;

Begin
  Initializeaza(dBinar1);
  Initializeaza(dBinar2);
                { aici se scrie cod pentru }
                { introducerea de date in  }
                { vectorii drum1 si drum2  }
  Codifica(drum1,dBinar1);
  Codifica(drum2,dBinar2);
  Intersectie(dBinar1,dBinar2,dBinar3);
  Writeln(DetNrLegaturi(dBinar3))
End.
```

Algoritmul Roy-Warshall

Fie G un graf neponderat și A matricea legăturilor directe între noduri (1 dacă două orașe sunt legate direct, 0 în caz contrar). Următorul algoritm calculează închiderea tranzitivă:

```
R:=A;
for k:=1 to N do
  for i:=1 to N do
    for j:=1 to N do
      if r[i,k]=1 and r[k,j]=1 then r[i,j]=1
```

Reprezentând matricea legăturilor directe sub forma descrisă la problema anterioară rezultă un algoritm elegant și extrem de rapid. Notăm prin R_i linia i (văzută ca un șir de biți):

```
R:=A;
for k:=1 to N do
  for i:=1 to N do
    if r[i,k]=1 then R[i]:=R[i] or R[k].
```

Programul este următorul (o linie a matricei drumurilor este o înșiruire de variabile de tip Word):

Listing: ROY.PAS

```
program WarshallBit;
uses Crt;
const MaxN=50;
type linie=array[1..MaxN div 16+1] of Word;
```

```
var a:array[1..MaxN] of linie;
    n,i,j,k,el,depl:Word;
    masca:Word; f:Text;
procedure Citire;
var c:Byte;
begin
  Assign(f,'GRAF.IN'); Reset(f);
  Readln(f,n);
  for i:=1 to n do
  begin
    for j:=1 to n do
    begin
      Read(f,c);
      if c=1
      then
      begin
        el:=j div 16+1;
        depl:=j mod 16;
        depl:=16-depl;
        masca:=1 shl depl;
        a[i,el]:=a[i,el] or masca
      end
    end;
  end;
  Close(f)
end;
procedure LogicalOr(Q,W:linie;
                    var E:linie);
begin
    for j:=1 to (n-1) div 16+1 do
      e[j]:=q[j] or w[j]
end;
procedure Warshall;
begin
  for k:=1 to n do
    for i:=1 to n do
    begin
      el:=k div 16+1;
      depl:=k mod 16;
      masca:=1 shl (16-depl);
      if a[i,el] and masca<>0
      then LogicalOr(a[i],a[k],a[i])
    end
end;

procedure Afisare;
begin
  for i:=1 to n do
  begin
    for j:=1 to n do
    begin
      el:=j div 16+1;
      depl:=j mod 16;
      masca:=1 shl (16-depl);
      if a[i,el] and masca<>0
      then Write(1,' ')
      else Write(0,' ')
    end;
    Writeln
  end
end;

Begin
  Clrscr;
  Citire;
  Warshall;
  Afisare
End.
```

Discuție:
Comparați viteza de lucru a acestui algoritm cu cel clasic.

Transfer de lungime

La problema determinării cuplajului într-un graf bipartit trebuie efectuată la un moment dat operația de transfer de lungime între un cuplaj deja construit și un lanț alternant crescător. Dacă cuplajul și lanțul alternant sunt reprezentate ca și la problema cu drumurile hamiltoniene, atunci operația de transfer de lungime `(K-L)∪(L-K)`, unde `K` este cuplajul, iar `L` este lanțul alternant crescător, se poate scrie astfel:

```
(K and not (K and L)) or (L and not (K and L))
```

unde `K` și `L` sunt reprezentările cuplajului și lanțului sub formă de matrice de biți.

Această expresie nu se poate simplifica deoarece operațiile se fac în raport cu graful `G`.

Jocul Nim

Se dau `n` grămezi de obiecte. Fiecare grămadă conține `a[i]` obiecte. Alternativ, doi jucători `A` și `B` extrag dintr-o singură grămadă un număr de obiecte (cel puțin unul). Cel care nu mai poate muta (adică toate obiectele au fost luate), pierde. Se cere să se determine dacă jucătorul `A` (care este primul la mutare) are o strategie sigură de câștig. În caz afirmativ să se programeze mutările lui `A`, cele ale lui `B` fiind citite de la intrare.

Soluție

O prezentare teoretică completă poate fi găsită în [1]. Aici voi puncta doar lucrurile legate de lucrul la nivel de bit.

Jucătorul `A` are strategie sigură de câștig doar dacă suma modulo `2` a reprezentărilor în baza `2` a numerelor `a[i]` este diferită de zero. Acest lucru îl calculăm folosind operatorul **xor**.

În acest caz jucătorul `A` va extrage dintr-o grămadă un număr de obiecte, astfel încât suma calculată în modul descris anterior să devină nulă. Acest lucru se face determinând din suma modulo `2` cea mai semnificativă poziție pe

care se află un 1 binar, iar apoi se complemetează toate cifrele din dreapta.

După efectuarea acestei mutări, suma modulo 2 va deveni 0, iar orice mutare efectuată de jucătorul B va trece jocul într-o poziție în care suma modulo 2 va fi diferită de zero.

Listing: JOC.PAS

```
program jocul;
const MaxN=5000;
var s,nr,n,i,suma,poz,gramada:Word;
    f:Text;
    a:array[1..MaxN] of Word;

function final:Boolean;
var suma,i:Word;
begin
  i:=1;
  while (i<=n) and (a[i]=0) do Inc(i);
  final:=i=n+1
end;

procedure Citire;
begin
  Assign(f,'JOC.IN'); Reset(f);
  Readln(f,n);
  suma:=0;
  for i:=1 to n do
  begin
    Read(f,a[i]);
    suma:=suma xor a[i]
  end;
  Close(f)
end;

Begin
  Citire;
  if suma=0
  then
  begin
    Write('A nu are strategie sigura');
    Writeln(' de castig.');
    Exit
  end;
  Write('A are strategie sigura');
  Writeln(' de castig');
  while not final do
  begin
    suma:=0;
    for i:=1 to n do
      suma:=suma xor a[i];
    poz:=Trunc((Ln(suma)/ln(2)));
    if 1 shl (poz+1)=suma
    then Inc(poz);
    i:=1;
    while (1 shl poz) and a[i]=0 do
      Inc(i);
    s:=suma xor a[i];
    Write('A extrage din gramada ');
    Writeln(i,' ',a[i]-s,'obiecte');
    a[i]:=s;
    suma:=0;
    if not(final)
    then
    begin
    Writeln('Introduceti mutarea lui B:');
      Readln(gramada,nr);
      a[gramada]:=a[gramada]-nr
    end
  end
End.
```

Reprezentarea submulțimilor

Se dă o submulțime a unei mulțimi de N elemente. Se cere să se reprezinte submulțimea dată cât mai eficient.

Soluție:

Acest tip de reprezentare îl întâlnim și la implementarea tipului **set** din Pascal. O submulțime a unei mulțimi 0..MaxN este un șir de biți, unde pe poziția k (în șirul de biți) avem valoarea 1 dacă elementul k aparține submulțimii și 0 în caz contrar. Se poate demonstra că există o bijecție între submulțimile unei mulțimi de N elemente și mulțimea șirurilor binare de lungime N.

Operațiile de intersecție, reuniune și diferență sunt realizate tot la nivel de bit. Se ține cont de faptul că în memorie octeții sunt numerotați crescător de la stânga la dreapta, iar biții în cadrul octetului sunt numerotați de la stânga la dreapta.

Listing: SUBMULT.PAS

```
program SubMultimi;
const MaxN=32;
type TMultime=array[1..MaxN] of Byte;
var i:Byte;
    m1,m2,m3:TMultime;

function Apartine(x:Byte;a:TMultime):Boolean;
var octet,poz:Byte;
begin
  octet:=x div 8+1;
  poz:=x mod 8;
  Apartine:=a[octet] and (128 shr poz)>0
end;

procedure Adauga(x:Byte; var a:TMultime);
var octet,poz:Byte;
begin
  octet:=x div 8+1;
```

```
  poz:=x mod 8
end;

procedure Intersectie(a,b:TMultime;
                      var c:TMultime);
begin
  for i:=1 to MaxN do
    c[i]:=a[i] and b[i]
end;

procedure Reuniune(a,b:TMultime;
                      var c:TMultime);
begin
  for i:=1 to MaxN do
    c[i]:=a[i] or b[i]
end;

procedure Diferenta(a,b:TMultime;
                      var c:TMultime);
begin
  for i:=1 to MaxN do
    c[i]:=a[i] and not (a[i] and b[i])
end;

procedure MultimeVida(var a:TMultime);
begin
  for i:=1 to MaxN do
    a[i]:=0
end;

Begin
  MultimeVida(m1);
  MultimeVida(m2);
  Adauga(3,m1);
  Adauga(8,m1);
  Adauga(56,m1);
  Adauga(5,m2);
  Adauga(3,m2);
  Adauga(56,m2);
  Intersectie(m1,m2,m3)
End.
```

Submulțimi de sumă dată

Se cere să se determine dacă o valoare dată k se poate scrie ca sumă de numere dintr-un vector dat. Fiecare număr din vector se poate folosi o singură dată.

Soluție

Se aplică programarea dinamică. Trebuie construit un vector c de valori booleene, în care pe poziția i, avem valoarea true doar dacă numărul i se poate scrie ca sumă din vectorul cu valorile date. Deoarece lucrăm doar cu valori booleene (true și false) nu este nevoie de un vector de octeți (deoarece o valoare booleană se reprezintă pe un octet) ci de un șir de biți. Relațiile de recurență pentru programarea dinamică le puteți deduce din programul de mai jos. Remarcați valorile mari ale lui k pentru care se poate calcula soluția, spre deosebire de varianta clasică.

Listing: SUMA.PAS

```
program SumaData;
uses Crt;
const MaxN=50;
var valori:array[1..MaxN] of Byte;
    c:array[1..MaxN div 8] of Byte;
    n,i,j,k,octet,poz:Byte;
    f:Text;

procedure Citire;
begin
  Assign(f,'TIR.IN');
  Reset(f);
  Readln(f,n);
  Readln(f,k);
  for i:=1 to n do
    if not SeekEoln(f)
    then Read(f,valori[i]);
  Close(f)
end;

Begin
  c[1]:=128;
  for j:=1 to n do
  begin
    for i:=0 to k-valori[j] do
    begin
      octet:=i div 8+1;
      poz:=i mod 8;
      if c[octet] and (128 shl poz)>0
      then
      begin
        octet:=(i+valori[j]) div 8+1;
        poz:=(i+valori[j]) mod 8;
        c[octet]:=c[octet] or 128 shl poz
      end
    end
  end;
  octet:=k div 8+1;
  poz:=k mod 8;
  if c[octet] and (128 shr poz)>0
  then Writeln('Posibil!')
  else Writeln('Imposibil!')
End.
```

Bibliografie:

1. Oltean Mihai - *Programarea jocurilor matematice*, Editura Albastră, Grupul Microinformatica, Cluj, 1996

NORTON CleanSweep
NORTON Utilities
NORTON AntiVirus
EasyRestore
PartitionMagic 4.0
LOST & FOUND
ROMSYM DATA-SRL
SOFTWARE
COREL DRAW 8
OMNILOGIC BGS
ALPIS
BRINEL
ALPIS®
Designed for Microsoft Windows NT Windows 98
WINDOWS 2000 READY PCs
ISO-9002
COMPAQ
P/N : 131015-a82
Deskpro EN SFF C466
6.4 GB Hard
64 MB RAM
NIC / Sound / W95
Monitor 15"
MAINBOARD
QDI WinneX 1, Intel 82810,
Intel Celeron 400PPGA W/128K cache
32 MB SDRAM DIMM PC 100 (8ns)
SVGA OB BOARD 4MB AGP
HDD 6.4 Quantum Fireball
CD-ROM 40x IDE
SOUND CARD on board +
Stereo Speakers 60W PMPO
FLOPPY 3.5", 1.44MB
ACER PS2 Keyboard WIN'95 +
Mouse SERIAL 3b + Pad
MONITOR COLOR
SVGA SMILE 14" DIGITAL
SALIENT
SALIENT Romania
MULTIMEDIA KIT
• CD-ROM CREATIVE LABS
• SOUND CARD 16bit
• BOXE 80W
SPRINT
MONITOR SPRINT 17" - model MP 1770N
PixelView
PixelView PlayTVpro
PV-BT878P+(W/FM)
Watch TV +Play TV+ + Motion Capture+
Teletext+ +Closed Capture+
FM Radio + + Remote Control+
Video Camera+ + Video Conferencing Ready
oferit de
ACER
THE COMPUTER COMPANY
Anul 2000 se lasă cu
PREMII
COMPUTER PRESS AGORA

Microsoft® România

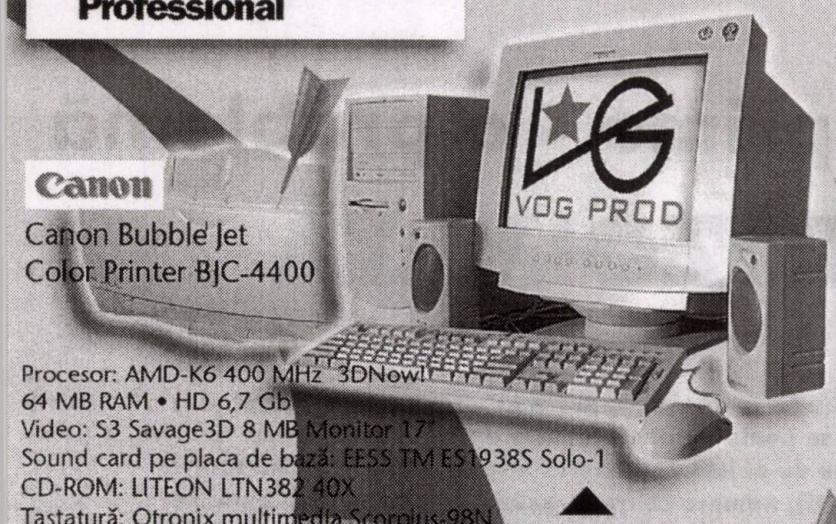
Microsoft Office 2000 Professional

TORNADO SISTEMS

Iomega Zip Drive
250 MB

Canon

Canon Bubble Jet
Color Printer BJC-4400



Procesor: AMD-K6 400 MHz 3DNow!
64 MB RAM • HD 6,7 Gb
Video: S3 Savage3D 8 MB Monitor 17"
Sound card pe placa de bază: EESS TM ES1938S Solo-1
CD-ROM: LITEON LTN382 40X
Tastatură: Qtronix multimedia Scorpius-98N
Mouse: Qtronix Lynx 3D
Modem: HSF 56k date/fax/voce intern, cu cipset Rockwell
Boxe: Mecmar LS-26A, 135W PMPO

GT Grup Transilvae

UMAX Astra 1220U
rezoluție optică 600x1200dpi
max. 9600x9600, col. 36bit, 1 pass

HEWLETT PACKARD
ROMÂNIA

HP DeskJet 610 C

Placa de bază ATX, INTEL 810, W/4 MB AGP&SOUND, UDMA 66
Procesor INTEL CELERON 400 (128kc) PPGA • Mémorie 32 MB SDRAM, PC-10
Video 4 MB AGP ON BOARD • Hard-disk QUANTUM 6.4 GB UDM
Tastatură GENIUS WIN 98 • Floppy 1.44 MB FDD • Mouse SCROLL
Monitor 14" SVGA LR NI DIGITAL MPRII • CD-ROM
Sound ON BOARD AC 97 AUDIO
Boxe 120 W, ACTIVE SPEAK S

ABONEAZĂ-TE și câștigă!

Abonează-te
la PCMagazine România,
PC Report sau Gazeta de informatică,
reviste editate de Computer Press Agora, până la data de 15 decembrie 1999.
Vei participa la tragerea la sorți care va desemna câștigătorii premiilor.
Trimite cărțile poștale din aceste reviste pe adresa:
Computer Press Agora, str.T.Vladimirescu nr.63/1, Tg.Mureș 4300 (CP 230/1)
Pentru informații suplimentare poți suna la: tel. 065-166516, fax 065-166290

Numere prime - o problemă încă NEREZOLVATĂ

Valentin Gheorghiță

Securitatea datelor - principala problemă apărută în ultimii ani în cadrul rețelelor de calculatoare - se poate rezolva în mare parte prin algoritmi criptografici, care au nevoie de obicei de generatoare de numere prime foarte mari (pe 128, 256 și 512 biți), numere ce nu se găsesc în tabelele matematice. Ca urmare este nevoie să se implementeze programe de generare a numerelor prime și de testare a primalității.

Cel mai răspândit algoritm de generare a numerelor prime este *Ciurul lui Eratostene*, dar acest algoritm este imposibil de implementat pentru numere mari din cauza lipsei memoriei (pentru generarea numerelor prime de 128 biți ar fi necesari 2^{105} Mb).

Cea mai cunoscută metodă de testare a primalității unui număr natural este împărțirea lui la toate numerele naturale mai mici sau egale cu radicalul lui. Această metodă este total ineficientă din cauza timpului necesar rulării ei (pentru un număr de 128 biți sunt necesari aproximativ 4358 ani pe un calculator care efectuează 134.000.000 operații pe secundă).

Totuși problema testării primalității numerelor mari este o problemă rezolvată în mare parte de algoritmii euristici, algoritmi care determină cu o precizie foarte mare dacă un număr este prim sau nu. Înainte de a prezenta câțiva dintre acești algoritmi este necesară prezentarea unor funcții cu viteză foarte mare care implementează câteva operații matematice cu numere mari. Aceste funcții primesc ca argument numere sub formă binară, reprezentate pe `n` biți, cea mai nesemnificativă cifră fiind cea cu indicele 0. Numerele furnizate ca răspuns de către funcții sunt numere binare reprezentate tot pe `n` biți. De asemenea, funcțiile setează, dacă este nevoie, valoarea indicatorului `carry` (indicator care indică dacă există depășire sau nu).

Funcția sumă

Funcția realizează *suma* a doi operanzi primiți ca parametri. De asemenea, funcția setează indicatorul `carry`.

```
sumă(op1,op2)
  carry=0
  pentru i de la 0 la n-1
    s[i]=op1[i]+op2[i]+carry
    carry=s[i]/2
    s[i]=s[i]%2
returnează s
```

Funcția dif

Funcția realizează *diferența* a doi operanzi primiți ca parametri (`op1-op2`).

```
dif(op1,op2)
  d=not(op2); d=sumă(d,1); d=sumă(op1,d)
returnează d
```

Funcția comp

Funcția realizează *compararea* a doi operanzi; ea returnează:

`-1` dacă `op1<op2`
`0` dacă `op1=op2`
`1` dacă `op1>op2`

```
comp(op1,op2)
  pentru i de la n-1 la 0
    dacă op1[i]>op2[i]
    atunci returnează 1
    altfel dacă op1[i]<op2[i]
         atunci returnează -1
returnează 0
```

Funcția mod

Funcția realizează *operația modulo* prin scăderi repetate.

```
mod(op1,op2)
  m=op1
  cât timp (comp(m,op2)>=0)
  execută m=dif(m,op2)
returnează m
```

Deoarece funcția implementează un algoritm prin scăderi repetate, este o foarte mare consumatoare de timp atunci când operanzii sunt diferiți ca ordin de mărime. În continuare vom prezenta un algoritm rapid ce rezolvă operația modulo.

```
fastmod(op1, op2)
  dacă (comp(op1,op2)>=0) şi (comp(op2,0)<>0)
  atunci tmp=op2
  p1=poz(op1)
  p2=poz(op2)
  tmp=shl(tmp,p1-p2-1)
  fm=mod(op1,tmp)
  dacă comp(fm,op2)>=0
  atunci fm=fastmod(fm,op2)
returnează fm
```

Funcţia `poz` returnează poziţia celui mai semnificativ bit al argumentului.

Funcţia `shl(x,y)` realizează deplasarea logică spre stânga a biţilor lui `x` cu `y` poziţii.

Funcţia amulb

Funcţia realizează *înmulţirea modulo* `p` a doi operanzi. Funcţia returnează rezultatul operaţiei: `(op1*op2)%p`.

```
amulb(op1,op2,p)
  rez=0
  k=not(p);
  k=sumă(k,1)
  pentru i de la 0 la n-1
    dacă op2(i)=1
      rez=sumă(rez,op1)
    cât timp carry=1 execută
      rez=sumă(rez,k)
    op1=shl(op1)
    cât timp carry=1 execută
      op1=sumă(op1,k)
    cât timp comp(rez,p)>=0 execută
      rez=sumă(rez,k)
returnează rez
```

Funcţia `shl` realizează deplasarea logică a biţilor la stânga, setând indicatorul `carry` dacă bitul care s-a pierdut în urma deplasării avea valoarea `1`.

Funcţia fastexp

Funcţia calculează *ridicarea la putere modulo* `p`. Funcţia returnează rezultatul operaţiei: $op1^{op2}\%p$

```
fastexp(op1,op2,p)
  exp=1
  cât timp compop(2,0)<>0 execută
    cât timp mod(op2,2)=0 execută
      op2=shr(op2)
      op1=amulb(op1,op1,n)
    op2=dif(op2,1)
    exp=amulb(exp,op1,n)
returnează exp
```

Funcţia cmmdc

Funcţia calculează *cel mai mare divizor comun* a două numere naturale.

```
cmmdc(a,b)
  dacă b=0 atunci returnează a
           altfel a=mod(a,b)
                  returnează cmmdc(b,a)
```

În continuare vă vom prezenta trei algoritmi care testează primalitatea unui număr. Aceşti algoritmi au o complexitate de ordinul *O(log n)*.

1. Testul Fermat

- se bazează pe teorema lui *Euler-Fermat*. Acestă teoremă spune că `p` este un număr prim dacă:

$a^{p-1}\%p=1$, *pentru orice număr întreg* $a\in[1,p)$.

Testul constă în alegerea unui `a` mai mic decât `p` şi acceptarea că `p` este prim dacă satisface condiţia de mai sus. În practică este necesară testarea doar a câtorva valori ale lui `a` şi nu a tuturor valorilor din intervalul `(1,p)`. Este recomandabil ca aproximativ o sută de teste să fie făcute pentru a fi convinşi că numărul `p` ales este prim.

2. Testul Solovay-Strassen

- identifică numerele *Carmichael composite*. Se alege aleator un număr $a\in(1,p-1)$ şi se testează dacă:

$cmmdc(a,p)=1$ şi $J(a,p)=a^{(p-1)/2}\%p$

unde `J(a,p)` este *operatorul Jacobi*.

Dacă `p` este prim atunci el verifică cele două condiţii, iar dacă el este composit, verificarea se face doar cu o probabilitate de `1/2`. Însă efectuând `k` teste, probabilitatea apariţiei unei erori scade la $1/2^k$.

Operatorul Jacobi este definit în felul următor:

$J(a,p)=1$ *dacă* $x^2\%p=a$ *are soluţie în inelul modulo* `p`, *adică* $a^{(p-1)/2}\%p=1$

$J(a,p)=-1$ *dacă* $x^2\%p=a$ *nu are soluţie în inelul modulo* `p`.

Funcţia care evaluează operatorul Jacobi este următoarea:

```
J(a,p)
  dacă a=1
  atunci J=1
  altfel
    dacă a%2=0
    atunci dacă (p*p-1)/8%2=0
           atunci J=J(a/2,p)
           altfel J=-J(a/2,p)
    altfel dacă (a-1)*(p-1)/4%2=0
           atunci J=J(p%a,a)
           altfel J=-J(p%a,a)
returnează J
```

Precizăm că în baza `p` numărul `-1` este egal cu `p-1`.

3. Testul Rubin

Pentru ca un număr `p` să fie prim, el trebuie să fie un întreg impar, deci poate fi reprezentat ca:

$p=2^r*s+1$

unde `s` este un întreg impar. Testul alege un număr de valori `a` în mod aleator din intervalul `(1,p-1)` şi se acceptă că `p` este prim dacă:

$a^s\%p=1$ sau $(a^{2^j})^s\%p=-1$ pentru $a<j<r$.

În toate celelalte cazuri `p` nu este prim.

Valentin Gheorghiţă este student în anul III la Universitatea Politehnică din Bucureşti. Poate fi contactat prin e-mail la adresa: septica@sundy.cs.ro.

CONCURS

Datorită faptului că numărul anterior GInfo a fost lansat pe piață după închiderea ediției pentru acest număr, rezolvările voastre nu au ajuns la redacție în timp util pentru a putea realiza un clasament intermediar. Vă reamintim că termenul limită pentru trimiterea problemelor din prima rundă este 15 Noiembrie 1999.

Termen

Rezolvările problemelor publicate în acest număr pot fi trimise până la data de:

15 Decembrie 1999

Hardware

Programele sunt rulate pe un calculator Pentium™ III la 450 MHz și 128 MB memorie internă. Se garantează că în momentul execuției programului vor fi disponibili cel puțin 300 KB de memorie convențională.

Software

Sistemul de operare folosit este Microsoft Windows '98 Second Edition. În cazul în care programul efectuează operații ilegale, concurentul va primi 0 puncte pentru testul respectiv.

Fișiere auxiliare

Nu pot fi folosite fișiere auxiliare trimise împreună cu sursa; în cazul în care programul creează fișiere auxiliare în timpul execuției, acestea vor avea obligatoriu extensia .TMP, vor fi create în directorul curent și vor fi șterse înaintea terminării execuției. Dacă această regulă nu este respectată concurentul va primi 0 puncte pentru problema respectivă chiar dacă programul furnizează soluția corectă. În cazul în care prin crearea de fișiere este pusă în pericol desfășurarea concursului, redacția poate decide descalificarea anumitor concurenți. Aceste noi reguli intră în vigoare începând cu runda a doua.

Soluții

În paginile GInfo vor apărea rezolvări corecte trimise de diferiți concurenți. Pentru ca o anumită rezolvare să fie publicată, fișierul trimis spre evaluare va trebui să conțină o descriere a metodei de rezolvare.

Timer

Nu este admisă modificarea timer-ului, orice tentativă în acest sens ducând la descalificarea concurentului.

Runda 2

CR2P1 Factorial

Dându-se un număr natural n să se determine numărul de cifre al lui n!. Numărul n (0≤n≤2.000.000.000) va fi citit din fișierul FACT.IN, iar rezultatul va fi scris în fișierul FACT.OUT. Programul va trebui să ruleze în cel mult o secundă.

Exemple

FACT.IN	FACT.OUT
5	3

FACT.IN	FACT.OUT
0	1

FACT.IN	FACT.OUT
8	5

CR2P2 Spiriduși

La un moment dat spiridușul MES s-a îndrăgostit de spiridușa GIS. Pentru a scăpa de insistențele spiridușului, spiridușa s-a refugiat în hiperspațiu. În fiecare zi GIS înaintează cu o unitate în sensul pozitiv al unei axe din hiperspațiu (axa poate fi schimbată în fiecare zi). Știind că hiperspațiul are n dimensiuni, că MES a aflat de dispariția lui GIS după s zile să se determine care este numărul p de puncte din hiperspațiu în care se poate afla GIS în momentul în care MES află de dispariția ei. Fișierul SPRITES.IN conține numerele n și s (1≤n, s≤20) separate printr-un singur spațiu. Fișierul SPRITES.OUT va trebui să conțină numărul p. Programul va trebui să ruleze în cel mult o secundă.

Exemple

SPRITES.IN	SPRITES.OUT
3 3	35

SPRITES.IN	SPRITES.OUT
4 4	126

CR2P3 Submulțime de sumă dată

Dându-se o mulțime A de n (1<n<10.000) numere reale și un alt număr real S, să se determine o submulțime B a mulțimii A cu proprietatea că suma elementelor mulțimii B este cât mai apropiată de S.

Date de intrare:

Fișierul SUMA.IN conține pe prima linie numărul n, pe a doua linie numărul S și pe următoarele n linii câte un număr real.

Date de ieșire:

Fișierul SUMA.OUT va avea o singură linie care conține indicii numerelor care fac parte din mulțimea B.

Exemple

SUMA.IN	SUMA.OUT
5 14 2.1 -0.3 4.1 5.2	1 2 4 5
7	

SUMA.IN	SUMA.OUT
5 6 1 1 1 1	1 2 3 4 5
1	

Timp maxim de execuție/test: 10 secunde.

Se acordă punctaje parțiale.

Neclarități

Dacă există neclarități în enunțurile problemelor, eventualele întrebări pot fi adresate redacției pe adresa **clara@cs.ubbcluj.ro**. Întrebările vor fi în așa fel formulate încât singurele răspunsuri posibile vor fi "DA" și "NU". Dacă această condiție nu este respectată sau răspunsul la întrebare reiese din enunț, răspunsul primit va fi "FĂRĂ COMENTARII". În cazul în care redacția consideră că întrebarea nu este corect formulată, răspunsul primit va fi "VĂ RUGĂM, REFORMULAȚI".

Grafică DIRECTCOLOR în Pascal

Florin Sabău

Într-un număr anterior al revistei "GInfo" a fost publicat un articol despre realizarea graficii în 256 de culori în Pascal. Continuăm în acest număr cu prezentarea modurilor grafice cu un număr superior de culori (modurile Hicolor pe 15, 16 biţi şi Truecolor pe 32 de biţi. Aceste moduri grafice se mai numesc şi Directcolor deoarece culoarea unui pixel nu este un index într-o paletă de culori (ca la modurile pe 16, 256 culori), ci este codificarea directă a componentelor fundamentale Red, Green şi Blue (roşu, verde şi albastru) ale acelei culori. La sfârşitul articolului veţi găsi implementate, într-un unit Pascal, funcţiile de bază necesare pentru a putea folosi modurile grafice Directcolor.

În ziua de astăzi există o gamă foarte largă de plăci grafice, fiecare având caracteristicile şi facilităţile sale proprii, ceea ce face ca programarea lor să fie foarte dificilă. De aceea, la începutul anilor '90, s-a încercat realizarea unei proceduri standard de programare a plăcilor grafice, aceeaşi pentru orice placă, şi uşor de implementat. Acest standard, realizat de *VESA* (*Video Electronics Standards Association*), a fost numit *Super VGA BIOS Extensions* (*VBE*). *VBE* pune la dispoziţia programatorului mai multe funcţii care returnează informaţii despre hardware-ul video din sistem, cum ar fi identificatorul plăcii grafice (Ex: *S3 Incorporated 86C775/86C785* pentru o placă S3), informaţii despre modurile grafice suportate, cantitatea de memorie video instalată etc. Articolul de faţă nu reprezintă o prezentare exhaustivă a standardului *VBE 2.0*, ci prezintă doar funcţiile absolut necesare pentru utilizarea modurilor grafice Directcolor.

Majoritatea plăcilor grafice suportă standardul *VBE*, versiunea 1.2, iar cu ajutorul unui update software şi versiunea 2.0, care aduce în plus câteva moduri grafice nestandard.

Funcţiile *VBE* pot fi accesate prin intermediul întreruperii *VIDEO BIOS* `10h`. Toate funcţiile returnează informaţii în registrul `AX` după cum urmează:

`AL=4Fh` dacă funcţia este suportată;
`AL≠4Fh` dacă funcţia nu este suportată;
`AH=00h` funcţia a fost apelată cu succes;
`AH=01h` apelul funcţiei a eşuat.

Funcţia 00h: Informaţii generale despre adaptorul grafic

Scopul acestei funcţii este de a pune la dispoziţia programului apelant informaţii referitoare la capabilităţile plăcii grafice. Datele returnate de funcţie sunt într-o zonă de 256 de octeţi specificată de utilizator.

La intrare:	
`AX`	`4F00h` *Funcţia 00h*
`ES:DI`	*segmentul şi offset-ul blocului de 256 de octeţi*
La ieşire:	
`AX`	`004Fh` *Dacă apelul funcţiei a avut succes*
	Informaţiile se află în zona corespunzătoare

Blocul de informaţii are structura următoare (scrisă sub forma unei înregistrări în Pascal):

```
type VGAInfoBlock=record
  VESA:array[0..3] of Char;
                { semnatura pe 4 octeti='VESA' }
  Ver:Word;
                { versiunea-ex. Hi(ver)=1 }
                { si Lo(ver)=2 pentru 1.2 }
  OEM:Longint;
              { pointer la identificatorul OEM }
              { al placii grafice              }
  Capabilities:Longint;
 { este 0 dacă DAC-ul este fixat la 6          }
 { bits/culoare primara, este 1 daca poate fi }
```

```
{ modificat numarul de bits/culoare, numai }
{ pentru moduri ce necesita paleta de culori }
 Modes:Longint;
              { pointer la lista modurilor }
              { grafice suportate          }
 VideoRAM:Word;
            { cantitatea de memorie video }
            { in blocuri de 64k           }
 Reserved: array[0..235]of Byte
                  { octeti rezervati pentru }
                  { dezvoltari ulterioare   }
end;
```

Câmpul `OEM` este un pointer spre un șir de caractere care identifică placa grafică.

Câmpul `Modes` este un pointer la o listă cu numerele modurilor grafice Super VGA suportate. Fiecare dintre acestea ocupă un cuvânt (16 biți), sfârșitul listei fiind reprezentat de un -1 (`FFFFh`).

Având această listă la dispoziție putem să aflăm informații specifice despre un anumit mod grafic prin intermediul funcției VBE `01h`.

Deoarece lista modurilor grafice poate fi stocată atât static, în memoria *ROM* (*Read Only Memory*), cât și generată dinamic la rulare în *RAM* (*Random Access Memory*), este de datoria aplicației apelante să testeze disponibilitatea unui mod grafic prin intermediul funcției VBE `01h`.

Funcția 01h: Informații despre modurile grafice suportate

Această funcție returnează într-o zonă definită de utilizator informații despre un anumit mod grafic, identificat prin numărul său găsit în lista de la funcția `00h`. Apelul funcției constă în încărcarea regiștrilor corespunzător cu tabelul de mai jos și executarea întreruperii `10h`:

La intrare:		
AX	4F01h	*Funcția 01h*
CX	*mod*	*unul dintre modurile grafice returnate de funcția 00h*
ES:DI	*segmentul și offset-ul blocului de 256 de octeți*	
La ieșire:		
AX	004Fh	*dacă apelul funcției a avut succes și modul grafic este disponibil*
		Informații în zona corespunzătoare

Blocul de informații este tot de 256 de octeți și are structura următoare:

```
type ModesInfo=record
  modul:Word;            { atributele modului }
  A,B:Byte;
       { atributele ferestrei A respectiv B }
  gran:Word;         { granularitatea ferestrei }
                     { in KB, de obicei 64KB    }
  size:Word;
                { marimea ferestrei în KB - de obicei 64KB }
  sega,segb:Word;          { segmentul de start }
                { pentru fereastra A respectiv B (A000h) }
  Bankswitch:pointer;  { pointer la functia }
                { de schimbare a bank-lui de lucru }
  octeti:Word;             { octeti pe linie de }
                { scanare=latime*nrbiti_per_pixel/8 }
  lat:Word;                { latimea in pixeli }
  inalt:Word;              { inaltimea in pixeli }
  lc:Byte;          { latime caracter in pixeli }
  li:Byte;
              { inaltimea unui caracter in pixeli }
  plane:Byte; { numar de plane de memorie=1 }
                { pentru moduri 15,16,32 biti }
  biti:Byte;               { numar de biti/pixel }
  bank:Byte; { numar de bank-uri de memorie }
  model:Byte;            { tip model de memorie }
  banks:Byte;              { marime bank in KB }
  pages:Byte;            { numar pagini imagine }
  rezervat:Byte;
  red,red1:Byte;           { marimea si pozitia }
          { mastii Red in octeții de culoare }
  green,green1:Byte;       { marimea si pozitia }
        { mastii Green in octetii de culoare}
  blue,blue1:Byte;         { marimea si pozitia }
        { mastii Blue in octetii de culoare }
  reze,reze1:Byte;         { marimea si pozitia }
   { mastii rezervate in octetii de culoare }
{ numai pentru moduri grafice pe 32 de biti }
  inf:Byte;        {informatii extinse despre }
                         { modurile Directcolor }
  rez:array[0..215] of Byte      { rezervati }
end;
```

Câmpul `Modul` este un cuvânt care conține informații importante despre modul grafic selectat, în timp ce câmpurile `A` și `B` se referă la atributele ferestrelor `A` și `B`. Codificarea pe biți a acestor câmpuri este prezentată în tabelul 1.

Câmpul `Model` conține tipul modelului de memorie folosit pentru modul grafic selectat. Acesta poate lua valorile: `00h` - text, `01h` - grafică CGA, `02h` - grafică HGC, `03h` grafică EGA pe 16 culori, `04h` - grafică cu pixeli împachetați, `05h` grafică "sequ 256", `06h` - grafică Directcolor (Hicolor, Truecolor), `07h` - YUV.

Pentru a înțelege semnificația celorlalte câmpuri, trebuie înțeles modul în care este organizată memoria video.

Memoria video este accesată prin intermediul adresei fizice `sega:0000h` (de obicei `A000h:0000h`), care se identifică cu pixelul din colțul stânga-sus. În funcție de modul grafic ales pixelul de coordonate (1, 0) poate fi modificat prin intermediul adresei `sega:0002h` pentru moduri pe 15, 16 biți respectiv `sega:0004h` pentru cele pe 32 de biți. Bineînțeles că necesitățile de memorie video pentru aceste moduri grafice sunt mult mai mari decât cei 64KB ai unui segment și de aceea memoria video este divizată în

bank–uri de cel mult 64Kb. Dimensiunea unui bank poate fi obținută prin consultarea câmpului `size`, aceasta fiind în majoritatea cazurilor de 64Kb. Pentru a schimba bank-ul curent și implicit pentru a putea scrie în altă zonă pe ecran se folosește funcția `BankSwitch`. Fiecărui pixel de pe ecran îi corespund în memoria video doi octeți (modurile Hicolor) sau patru octeți (modurile Truecolor) care reprezintă culoarea respectivului pixel. Acești octeți conțin codificarea pe biți a măștilor de culoare **Red**, **Green** și **Blue** din care este compusă culoarea (tabelul 2).

Tabelul 1

Informații despre mod grafic		Ferestrele A și B	
Bit	**Descriere**	**Bit**	**Descriere**
0	Modul este suportat	0	Setat dacă există
1	Informații opționale disponibile	1	Setat dacă suportă citirea
2	Suportă ieșire BIOS	2	Setat dacă suportă scrierea
3	Color (1) sau monocrom (0)	3-7	Rezervați
4	Mod grafic (1) sau mod text (0)		
5-15	Rezervați		

Această codificare este regăsită practic la toate plăcile grafice. Pentru a fi însă siguri, putem folosi câmpurile `Red`, `Red1`, `Blue`, `Blue1`, `Green`, `Green1` care conțin mărimea și respectiv poziția în octeții de culoare a componentelor Red, Green și Blue (roșu, verde și albastru).

Funcția 02h - Inițializare mod grafic Super VGA

Funcția `02h` inițializează modul grafic dorit. O listă a modurilor grafice disponibile poate fi consultată prin intermediul funcției `00h`. Apelul funcției constă în încărcarea regiștrilor după cum urmează și executarea întreruperii `10h`:

La intrare:

AX	4F02h	*Funcția 02h*
BX	mod	*unul dintre modurile grafice returnate de funcția 00h*

La ieșire:

AX	004Fh	*dacă apelul funcției a avut succes și modul grafic este disponibil*

Implementarea unei proceduri Putpixel

Toate cele trei funcții *VBE* de mai sus sunt bune și frumoase, dar fără o procedură de tip "putpixel", sunt complet nefolositoare. Pentru a scrie un pixel pe ecran sunt necesari doi pași:

1. Poziționarea pe bank-ul de lucru corect;
2. Scrierea propriu-zisă în memoria video a culorii dorite.

Numărul bank-ului se găsește cu formula:

(x**octeți_per_culoare* + y**octeți_per_linie_de_scanare*) **div** (*dimensiune_bank*),

unde *octeți_per_culoare* poate lua valorile 2 (moduri pe 15, 16 biți) sau 4 (moduri pe 32 de biți).

Pentru a găsi offsetul, în bank-ul selectat la pasul 1, unde se vor scrie octeții de culoare se folosește formula:

(x**octeți_per_culoare* + y**octeți_per_linie_de_scanare*) **mod** (*dimensiune_bank*).

O situație specială este aceea a modurilor pe 32 de biți în care o culoare este codificată pe 4 octeți. Dacă dimensiunea unui bank nu este un multiplu de 4, atunci acești octeți pot fi în bank-uri diferite (consecutive). În acest caz trebuie apelată de două ori procedura de setare a bank-ului (`setbank`) și scrierea separată a word-ului mai puțin semnificativ în primul bank și a celui mai semnificativ în următorul bank.

Unit-ul VBE

Unit-ul conține un număr de 13 proceduri și funcții. Urmează o scurtă descriere a acestora:

- `InfoVBE` - folosește funcția VBE `00h` pentru a tipări pe ecran informații despre placa grafică;
- `Info(mode:Word;var a:vesa;x:Boolean)` - folosește funcția VBE `01h` pentru a returna în variabila `a` informații despre modul grafic `mode`. Dacă parametrul `x` este `true`, va tipări aceste informații și la consolă;
- `Baza(n:Longint;b,biti:Integer):string` - convertește valoarea `n` în baza `b` și returnează rezultatul sub forma unui string;
- `SetSVGA(mode:Word)` - inițializează modul grafic `mode`;
- `Settext` - se revine la modul text;
- `PutPixel(x,y:Word;c:Longint)` - desenează un pixel de culoare `c`, în punctul de coordonate `(x,y)`;
- `Put(x,y:Word; r,g,b:Byte)` - la fel ca și `PutPixel`, dar culoarea este dată sub forma componentelor Red, Green și Blue;
- `RGB(r,g,b:Byte):Longint` - convertește valorile componentelor Red, Green și Blue la culoarea aferentă;

Tabelul 2

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mod grafic pe 15 biți:	0	R	R	R	R	R	G	G	G	G	G	B	B	B	B	B
mod grafic pe 16 biți:	R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B

mod grafic pe 32 biți:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B

- Setbank(l:Longint) - folosită de procedura PutPixel pentru a schimba bank-ul de lucru curent;
- Getmaxcolor - returnează culoarea maximă pentru modul grafic curent;
- Maxx, Maxy - returnează rezoluția pe orizontală respectiv pe verticală pentru modul grafic curent;
- Line(xs,ys,xe,ye:Integer;cul:Longint) - trasează o linie între punctele de coordonate (xs,ys) și (xe,ye).

Listing: VBE20.PAS

```
unit VBE;
interface
uses Crt,Dos;
type vesa=record
       modul:Word;
       a,b:Byte;
       gran:Word;
       size:Word;
       sega,segb:Word;
       bankfunc:pointer;
       octeti:Word;
       lat,inalt:Word;
       lc,ic,plane,biti,bank,model:Byte;
       rezerv,red,red1,green,green1:Byte;
       pages,banks,blue,blue1,reze:Byte;
       reze1,inf:Byte;
       rez:array[0..215] of Byte
     end;
type vgainfo=record
       vesa:array[0..3] of Char;
       ver:Word;
       oem:^Char;
       capabilities:Longint;
       modes:^Word;
       videoram:Word;
       reserved:array[0..235] of Byte
     end;
var s,o,cbank:Word;
    r:Registers;
    e:Boolean;
    bankswitch:pointer;
    i,j:Integer;
    a:vesa;
    b:vgainfo;
    id:string;

procedure infoVBE;
procedure info(mode:Word;var a:vesa;
                                 x:Boolean);
function baza(n:Longint;
                    b,biti:Integer):string;
procedure setsvga(mode:Word);
procedure settext;
procedure putpixel(x,y:Word; c:Longint);
procedure put(x,y:Word; r,g,b:Byte);
procedure setbank(l:Longint);
function getmaxcolor:Longint;
function maxx:Word;
function maxy:Word;
procedure line(xs,ys,xe,ye:Integer;
                                 cul:Longint);
function rgb(r,g,b:Byte):Longint;

implementation

function baza;
var s,t:string;
    i:Integer;
begin
  s:=''; t:='';
  for i:=1 to biti do
  begin
    if n mod b<10
    then s:=s+Chr(Ord('0')+n mod b)
    else s:=s+Chr(Ord('A')+n mod b-10);
    n:=n div b
  end;
  for i:=1 to Length(s) do
    t:=t+s[Length(s)-i+1];
  if b=16 then t:=t+'h';
  baza:=t
end;

procedure info;
begin
  e:=true;
  s:=Seg(a);
  o:=Ofs(a);
  r.ax:=$4F01;
  r.cx:=mode;
  r.es:=s;
  r.di:=o;
  Intr($10,r);
  if (r.al<>$4F) or (r.ah<>0)
  then begin e:=false; Exit end;
  if x then
    with a do
    begin
      Clrscr;
      Gotoxy((80-27) div 2,wherey);
  Write('Informatii despre modul ');
  Writeln(baza(r.cx,16,3));
  Write('Registrii la iesire AL= ');
  Write(baza(r.al,16,2));
  Writeln(' AH = ',baza(r.ah,16,2));
  Write('Atributele modului: ',modul);
  Writeln(' = ',baza(modul,2,8));
  Writeln('Atributele ferestrei A: ');
  Writeln(a,' = ',baza(a,2,8));
  Write('Atributele ferestrei B: ');
  Writeln(b,' = ',baza(b,2,8));
  Write('Granularitatea ferestrei: ');
  Writeln(gran,'KB');
  Write('Adresabilitatea ferestrei: ');
  Writeln(,size,'KB');
  Write('Start segment A: ',sega,' = ');
  Writeln(baza(sega,16,4));
  Writeln('Start segment B: ',segb,' = ');
  Writeln(baza(segb,16,4));
  Write('BankSwitch: ',seg(bankfunc^),':')
  Write(ofs(bankfunc^),'=');
```

```
  Writeln(baza(seg(bankfunc^),16,4));
  Writeln(':',baza(ofs(bankfunc^),16,4));
  Write('Octeti per linie de scanare:');
  Writeln(octeti);
  Write('Latime in pixeli grafic/text: ');
  Writeln(lat,' inaltime ',inalt);
  Writeln('Latime caracter:',lc,'inaltime',ic);
  Writeln('Plane de memorie: ',plane);
  Writeln('Biti/pixel: ',biti);
  Writeln('Bank-uri: ',bank);
  Write('Tip model de memorie: ');
  Write(model,' = ',baza(model,16,2));
  Writeln('Marime bank: ',banks,'KB');
  Writeln('Nr. pagini imagine: ',pages);
  Write('Marime masca RED: ',red);
  Writeln(' Marime camp RED: ',red1);
  Write('Marime masca GREEN: ');
  Writeln(green,'Marime camp GREEN: ',green1);
  Write('Marime masca BLUE: ');
  Writeln(blue,' Marime camp BLUE: ',blue1);
  Write('Marime masca: ');
  Writeln(reze,' Pozitie masca: ',reze1);
  Writeln('Informatii despre moduri: ',inf);
      Readkey
    end;
    bankswitch:=a.bankfunc
end;

procedure setsvga;
begin
  info(mode,a,false);
     { Se umple variabila a cu informații }
  asm                { despre modul selectat }
    mov ax,4f02h
    mov bx,mode
    int 10h
  end
end;

procedure setbank;
var b:Word;
begin
  b:=1*(64 div a.gran) shr 16;
        { Se calculează numărul bank-ului }
  if b<>cbank
  then
  begin
    cbank:=b;
    asm
      xor bx,bx    { BX=Numarul ferestrei A }
      mov dx,b     { sau B DX=bank-ul       }
      call[bankswitch]
    end
  end
end;

procedure settext; assembler;
asm
  mov ax,0003h; int 10h
end;
function getmaxcolor;
begin
 if a.biti<>32
 then getmaxcolor:=Trunc(Exp(a.biti*ln(2)))-
1
 else getmaxcolor:=16777215
end;

function maxx;
begin
  maxx:=a.lat-1
end;

function maxy;
begin
  maxy:=a.inalt-1
end;

procedure line;      { Algoritmul Bresenham
}
var dx,dy,iy,i,s:Integer;
begin
  dx:=xe-xs; dy:=ye-ys;
  if dy>=0 then iy:=1
           else iy:=-1;
  dy:=Abs(dy);
  if dx<0
  then
  begin xs:=xe; ys:=ye; dx:=-dx; iy:=-iy
end;
  if dx>dy
  then
  begin
    s:=-dx; dx:=dx*2; dy:=dy*2;
    for i:=0 to dx div 2 do
    begin
      putpixel(xs,ys,cul);
      Inc(xs); Inc(s,dy);
      if s>=0
      then begin Inc(ys,iy); Dec(s,dx) end
    end
  end
  else
  begin
    s:=-dy; dx:=dx*2; dy:=dy*2;
    for i:=0 to dy div 2 do
    begin
      putpixel(xs,ys,cul);
      Inc(ys,iy); Inc(s,dx);
      if s>=0
      then begin Inc(xs); Dec(s,dy) end
    end
  end
end;

procedure putpixel;{ Scrie un pixel pe ecran
}
var k,l:Longint; m,z:Word;
begin
  case a.biti of
    15,16: begin
             l:=Longint(a.octeti)*y+x*2;
```

```
            setbank(l);
            memw[a.sega:Word(l)]:=Word(c)
          end;
    32: begin
          l:=Longint(a.octeti)*y+x*4;
          k:=Longint(a.gran)*1024;
          z:=Word(l);
          setbank(l);
          if z<k-2
          then meml[a.sega:z]:=c
          else
          begin
            memw[a.sega:z]:=Word(c);
            setbank(l+k);
            memw[a.sega:0]:=c div $10000
          end
        end
  end
end;

procedure put;
var c:Longint;
begin
  c:=rgb(r,g,b);
  putpixel(x,y,c);
end;

function rgb;
    { Converteste componentele Red, Green }
    {  si Blue la culoarea respectiva     }
var c:Longint;
begin
  c:=0;
  case a.biti of
  15:c:=Word(r) shl 10+Word(g) shl 5+b;
  16:c:=Word(r) shl 11+Word(g) shl 5+b;
  24,32:c:=Longint(r) shl 16+Word(g) shl 8+b
  end;
  rgb:=c
end;

procedure infoVBE;
     { Informatii despre adaptorul grafic }
begin
  asm
    mov ax,4f00h
    mov di,offset b
    int 10h
  end;
  with b do
  begin
    id:=vesa[0]+vesa[1]+vesa[2]+vesa[3];
    Writeln('VESA ID.: ',id);
    Writeln('Versiune:
',Hi(ver),'.',Lo(ver));
    Write('Sir OEM: ');
    while oem^<>#0 do
    begin
      Write(oem^);
      Inc(oem)
    end;
    Write('Capabil.: ');
    Writeln(baza(capabilities,2,8));
    Writeln('Video RAM: ',videoram*64,'Kb');
    i:=0;
    while modes^<>$ffff do
    begin
      Write('Modul ',i:2,': ');
      Writeln(baza(modes^,16,4));
      Inc(modes,1);
      Inc(i);
      if Readkey=#27 then Break
    end
  end
end;

End.
```

Un posibil program de testare a unit-ului ar putea fi următorul:

Listing: TEST.PAS

```
Program test;
uses vbe20,crt;
var i,j:Word;
    maxcolor:Longint;
    a:modesinfo;
begin
  infoVBE;
  info($112,a,true);
  setsvga($112);
  maxcolor:=getmaxcolor;
  for i:=0 to maxx do
  for j:=0 to maxy do
    putpixel(i,j,(i*j) mod (maxcolor+1))
  Readkey;
  settext
End.
```

Plecând de la implementarea procedurii de scriere a unui pixel pe ecran, se poate îmbunătăți unit-ul și cu alte proceduri specifice unui unit de grafică:

- `getpixel` (procedură similară cu `putpixel`, dar în loc să se scrie în memoria video, se citesc informații de culoare);
- `rectangle`, `circle`, `arc`, `ellipse`, `bar` (au fost prezentate în [1]);
- `getimage`, `putimage` (se are în vedere salvarea informațiilor de culoare din mai multe bank-uri de lucru).

Bibliografie

1. Horea Haitonic - *Grafică în 256 de culori*, Gazeta de Informatică 4/1999
2. PC Report - *Standardul VESA Bios Extension*, martie 1998

Florin Sabău este elev în clasa a XI-a la Liceul "Traian Lalescu" din Reșița. Poate fi contactat prin e-mail la adresa: fsabau@cs.ro.

Pagini WEB cu JavaScript

Mihai Scorțaru

Prezentarea limbajului JawaScript continuă cu descrierea modului în care pot fi folosiți vectorii pentru a ușura proiectarea paginilor Web. De asemenea vor fi prezentate proprietățile și metodele obiectului Math și ale clasei String.

Vectori

Un vector (șir) este o structură folosită pentru a păstra date care au o anumită legătură unele cu altele. În *JavaScript* vectorii au o natură ***asociativă***, adică o anumită locație este asociată cu o anumită informație. În continuare vă vom prezenta câteva tipuri de vectori *JavaScript* precum și modul în care aceștia pot fi folosiți.

Vectori predefiniți

După cum am spus în episodul anterior este posibil ca anumite elemente HTML să fie tratate ca obiecte. Limbajul *JavaScript* permite, de asemenea, tratarea unor obiecte ca fiind vectori formați din alte obiecte. În figura 1 puteți vedea un exemplu de șiruri predefinite:

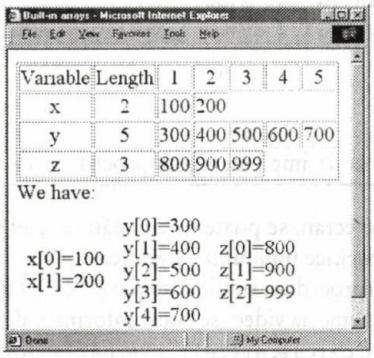


Fig. 1 Șiruri predefinite

Vectorul de ancore

Fiecare document HTML are asociat un șir special ale cărui componente sunt obiecte care desemnează ancorele prezente în respectivul document. Ordinea elementelor vectorului este ordinea apariției ancorelor în documentul HTML. În *JavaScript*, identificatorul acestui șir special este `anchors`. Să considerăm următorul exemplu:

```
<HTML>
<HEAD> <TITLE> Anchors Array </TITLE> </HEAD>
<BODY>
  <A href="http://www.ginfo.ro">
  Gazeta de informatică</A><BR>
  <A href="http://www.agora.ro">
  Computer PRESS Agora</A><BR>
  <A href="http://www.pcmagazine.ro">
  PC Magazine</A><BR>
  <A href="http://www.pcreport.ro">
  PC Report</A><BR>
  <A href="http://www.pcconcrete.ro">
  PC Concrete</A><BR>
  <A href="http://www.aol.ro">
  Agora On Line</A><BR>
</BODY>
</HTML>
```

În cazul documentului anterior vom avea un vector care va avea caracteristici similare cu vectorul `anchors` de mai jos:

```
anchors.length=6
anchors[1].value = http://www.ginfo.ro
anchors[2].value = http://www.agora.ro
anchors[3].value = http://www.pcmagazine.ro
anchors[4].value = http://www.pcreport.ro
anchors[5].value = http://www.pcconcrete.ro
anchors[6].value = http://www.aol.ro
```

Vectorul de formulare

La fel ca și în cazul ancorelor, există un șir care conține obiecte ce descriu formularele definite în cadrul unui document HTML. Ordinea elementelor vectorului este, și în acest caz, ordinea apariției formularelor în cadrul documentului HTML. Identificatorul corespunzător acestui vector este `forms`. Să considerăm exemplul:

```
<HTML>
<HEAD>
  <TITLE> Forms Array </TITLE>
</HEAD>
<BODY>
  <FORM name="Primul_formular">
  <!-- conținutul primului formular -->
  </FORM>
  <!-- ... -->
```

```
  <FORM name="Al_doilea_formular">
<!-- conținutul celui de-al 2-lea formular -->
  </FORM>
</BODY>
</HTML>
```

În acest caz vom avea `document.forms.length=2`, construcția `document.forms[1]` va fi echivalentă cu construcția `document.Primul_formular`, iar construcția `document.forms[2]` va fi echivalentă cu construcția `document.Al_doilea_formular`.

În continuare vom prezenta un exemplu de utilizare a vectorului `forms`. Să presupunem că aveți o pagină Web care conține mai multe formulare, și alături de butoanele **Submit** corespunzătoare fiecărui formular, doriți să creați un buton a cărui apăsare să determine transmiterea informațiilor cuprinse în toate formularele care apar în acea pagină. Codul *JavaScript* necesar pentru a realiza acest lucru este:

```
for x=1 to forms.length
  submit form[x]
next
```

Șiruri definite de către utilizator

În *JavaScript*, un vector este considerat și el ca fiind un obiect. Datorită faptului că, în cazul vectorilor definiți de către utilizator, este creat un nou obiect, acest nou obiect trebuie declarat, iar valorile sale trebuie inițializate. Următorul exemplu prezintă o funcție care poate fi folosită pentru inițializarea unui vector definit de utilizator:

```
function initArray(){
  this.length=initArray.arguments.length
  for (var i=1;i<=this.length;i++)
    this[i]=initArray.arguments[i-1]
}
```

Pentru a inițializa un anumit vector veți putea folosi o construcție de genul:

```
var newArray=new initArray("GInfo",
"PC Report","PC Magazine")
```

A fost creat un vector similar cu vectorul `newArray` de mai jos:

```
newArray.length=3
newArray[1]="GInfo"
newArray[2]="PC Report"
newArray[3]="PC Magazine"
```

Pentru a crea un vector cu elemente neinițializate funcția corespunzătoare va fi:

```
function emptyArray(){
  this.length=emptyArray.arguments.length
  for (var i=1;i<=this.length;i++) this[i]=""
}
```

Pentru a crea un vector cu elemente neinițializate veți putea folosi o construcție de genul:

```
  var newArray=new emptyArray(3)
```

A fost creat un vector similar cu vectorul `newArray` de mai jos:

```
newArray.length=3
newArray[1]=NULL
newArray[2]=NULL
newArray[3]=NULL
```

În următorul exemplu vom construi un șir care conține numele celor 41 de președinți ai Statelor Unite ale Americii. Într-o casetă text va putea fi introdus un număr, iar într-o altă casetă va apărea numele președintelui corespunzător.

```
<HTML>
<HEAD>
<TITLE> The Presidents of the United States of
        America
</TITLE>
  <SCRIPT language="JavaScript">
  function initArray(){
    this.length = 42
    this[0] = "  George Washington   "
    this[1] = "       John Adams     "
    this[2] = "    Thomas Jefferson  "
    this[3] = "     James Madison    "
    this[4] = "     James Monroe     "
    this[5] = "  John Quincy Adams   "
    this[6] = "    Andrew Jackson    "
    this[7] = "   Martin Van Buren   "
    this[8] = "William Henry Harrison "
    this[9] = "       John Tyler     "
    this[10] = "    James K. Polk     "
    this[11] = "    Zachary Taylor    "
    this[12] = "   Millard Fillmore   "
    this[13] = "   Franklin Pierce    "
    this[14] = "    James Buchanan    "
    this[15] = "   Abraham Lincoln    "
    this[16] = "    Andrew Johnson    "
    this[17] = "   Ulysses S. Grant   "
    this[18] = " Rutherford B. Hayes  "
    this[19] = "  James A. Garfield   "
    this[20] = "  Chester A. Arthur   "
    this[21] = "  Grover Cleveland    "
    this[22] = "  Benjamin Harrison   "
    this[23] = "  Grover Cleveland    "
    this[24] = "  William McKinley    "
    this[25] = "  Theodore Roosevelt  "
    this[26] = "   William H. Taft    "
    this[27] = "   Woodrow Wilson     "
    this[28] = "  Warren G. Harding   "
    this[29] = "   Calvin Coolidge    "
    this[30] = "   Herbert Hoover     "
    this[31] = "Franklin D. Roosevelt "
    this[32] = "   Harry S. Truman    "
    this[33] = " Dwight D. Eisenhower "
    this[34] = "    John F. Kennedy   "
    this[35] = "  Lyndon B. Johnson   "
    this[36] = "   Richard M. Nixon   "
    this[37] = "    Gerald R. Ford    "
    this[38] = "     Jimmy Carter     "
```

```
    this[39] = "     Ronald Reagan      "
    this[40] = "      George Bush       "
    this[41] = "      Bill Clinton      "
  }
  var presidents = new initArray()
  </SCRIPT>
</HEAD>
<BODY>
  <FORM name="list">
  The <INPUT type="text" name="nr" size="1"
  value="42" onchange=
  "document.list.pres.value=
  pr[document.list.nr.value-1]">-th  american
  president was <INPUT name="pres"
  type="text" value="-None-" disabled>.
  </FORM>
</BODY>
</HTML>
```

În figura 2 puteți vedea o captură a ferestrei programului de navigare *Internet Explorer 5.0* în timpul afișării paginii Web corespunzătoare documentului HTML anterior:

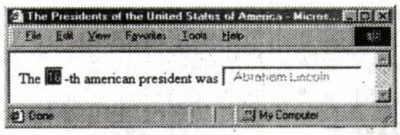


Fig. 2 Exemplu pentru șiruri definite de utilizatori

Operații matematice

Limbajul *JavaScript* include metode corespunzătoare mai multor operații matematice. Aceste metode permit efectuarea de operații simple fără a mai fi nevoie de realizarea unei interacțiuni cu serverul. Metodele necesare efectuării operațiilor matematice pot fi accesate prin intermediul obiectului `Math`.

Funcția abs()

Funcția `abs()` este folosită pentru a determina valoarea absolută a unui anumit număr; funcția va avea ca parametru numărul al cărui modul dorim să îl aflăm. Următorul exemplu ilustrează modul de folosire a acestei metode:

```
<HTML>
<HEAD> <TITLE> abs() </TITLE> </HEAD>
<BODY>
  <FORM name="absform">
  The absolute value of <INPUT type="text"
  name="nr" size="5" value="" onchange=
  "document.absform.absnr.value=
  Math.abs(document.absform.nr.value)"> is
  <INPUT name="absnr" type="text" size="5"
  value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

În figura 3 puteți vedea efectul obținut prin încărcarea documentului HTML anterior.

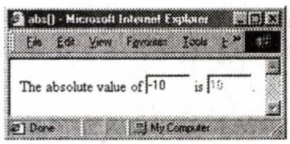


Fig. 3 Exemplu de utilizare a funcției `abs()`

Funcția ceil()

Funcția `ceil()` poate fi folosită pentru determinarea părții întregi superioare a unui număr care este transmis ca parametru. Partea întreagă superioară a unui număr x este definită ca fiind cel mai mic număr întreg mai mare sau egal cu numărul x dat. Modul de utilizare a acestei metode este ilustrat în următorul exemplu:

```
<HTML>
<HEAD> <TITLE> ceil() </TITLE> </HEAD>
<BODY>
  <FORM name="ceilform">
  The smallest integer number greater or
  equal to <INPUT type="text" name="nr"
  size="5" value="" onchange="document.ceil-
  form.ceilnr.value =
  Math.ceil(document.ceilform.nr.value)"> is
  <INPUT name="ceilnr" type="text" size="5"
  value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

Efectul obținut prin încărcarea documentului HTML anterior poate fi observat în figura 4.

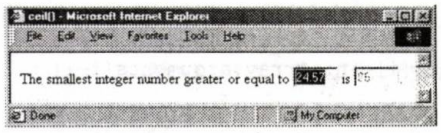


Fig. 4 Exemplu de utilizare a funcției `ceil()`

Funcția exp()

Funcția `exp()` este folosită pentru a calcula valoarea e^x, unde x este un număr transmis ca parametru. În următorul exemplu puteți observa modul de utilizare a acestei funcții.

```
<HTML>
<HEAD>
  <TITLE> exp() </TITLE>
</HEAD>
<BODY>
  <FORM name="expform">
    e^<INPUT type="text" name="nr" size="5"
    value="" onchange=
    "document.expform.expnr.value =
    Math.exp(document.expform.nr.value)"> =
    <INPUT name="expnr" type="text" size="5"
    value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

În figura 5 poate fi observat efectul obținut prin încărcarea documentului HTML anterior.

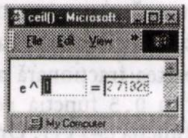


Fig. 5 Exemplu de utilizare a funcției `exp()`

Funcția floor()

Funcția `floor()` este funcția pereche pentru `ceil()`. Ea poate fi folosită pentru determinarea părții întregi inferioare a unui număr care este transmis ca parametru. Partea întreagă inferioară a unui număr x este definită ca fiind cel mai mare număr întreg mai mic sau egal cu numărul x dat. Modul de utilizare a acestei metode este ilustrat în următorul exemplu:

```
<HTML>
<HEAD> <TITLE> floor() </TITLE> </HEAD>
<BODY>
  <FORM name="floorform">
    The greatest integer number snaller or
    equal to <INPUT type="text" name="nr"
    size="5" value="" onchange=
    "document.floorform.floornr.value=
    Math.floor(document.floorform.nr.value)">
    is <INPUT name="floornr" type="text"
    size="5" value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

Efectul obținut poate fi observat în figura 6.

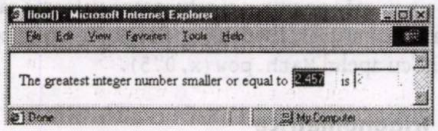


Fig. 6 Exemplu de utilizare a funcției `floor()`

Funcția log()

Funcția pereche pentru `exp()` este `log()`; cu ajutorul acestuia poate fi calculat logaritmul natural al unui număr transmis ca parametru. Un exemplu de utilizare a acestei funcții este:

```
<HTML>
<HEAD> <TITLE> log() </TITLE> </HEAD>
<BODY>
  <FORM name="logform">
    ln <INPUT type="text" name="nr" size="5"
    value="" onchange=
    "document.logform.lognr.value =
    Math.log(document.logform.nr.value)"> =
    <INPUT name="lognr" type="text" size="5"
    value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

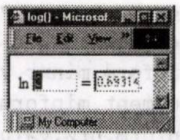


Fig. 7 Exemplu de utilizare a funcției `log()`

Funcția max()

Această funcție are ca parametri două numere și returnează valoarea celui mai mare dintre ele. Următorul exemplu ilustrează modul de utilizare a acestei funcții.

```
<HTML>
<HEAD> <TITLE> max() </TITLE> </HEAD>
<BODY>
  <FORM name="maxform">
  The greatest number between
  <INPUT type="text" name="nr1" size="5"
    value="" onchange=
    "document.maxform.maxnr.value =
    Math.max(document.maxform.nr1.value,
    ocument.maxform.nr2.value)"> and
  <INPUT type="text" name="nr2"
    size="5" value=""
    onchange="document.maxform.maxnr.value =
    Math.max(document.maxform.nr1.value,
    document.maxform.nr2.value)"> is
    <INPUT name="maxnr" type="text" size="5"
    value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

După încărcarea documentului HTML anterior, fereastra programului de navigare ar putea arăta ca în figura 8.

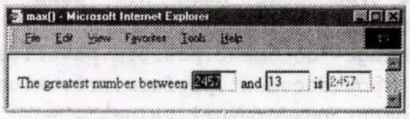


Fig. 8 Exemplu de utilizare a funcției `max()`

Funcția min()

Funcția `max()` are și ea o funcție pereche, și anume `min()`. Această funcție are tot două numere ca parametri și returnează valoarea celui mai mic dintre ele. Un exemplu de utilizare este următorul:

```
<HTML>
<HEAD>
  <TITLE> min() </TITLE>
</HEAD>
<BODY>
  <FORM name="minform">
  The smallest number between
  <INPUT type="text" name="nr1" size="5"
    value="" onchange=
    "document.minform.minnr.value=
    Math.min(document.minform.nr1.value,
    document.minform.nr2.value)"> and
```

```
    <INPUT type="text" name="nr2" size="5"
    value="" onchange=
    "document.minform.minnr.value =
    Math.min(document.minform.nr1.value,
    document.minform.nr2.value)"> is
  <INPUT name="minnr" type="text" size="5"
  value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

Fereastra programului de navigare ar putea arăta ca în figura 9.

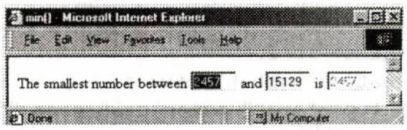


Fig. 9 Exemplu de utilizare a funcției `min()`

Funcția pow()

Limbajul *JavaScript* pune la dispoziție funcția `pow()` care poate fi folosită pentru ridicarea la putere. Această funcție are doi parametri, primul fiind baza, iar al doilea exponentul. Un exemplu de utilizare este prezentat în continuare.

```
<HTML>
<HEAD> <TITLE> pow() </TITLE> </HEAD>
<BODY>
  <FORM name="powform">
  <INPUT type="text" name="nr1" size="5"
    value="" onchange=
    "document.powform.pownr.value =
    Math.pow(document.powform.nr1.value,
    document.powform.nr2.value)"> ^
  <INPUT type="text" name="nr2" size="5"
    value=""
    onchange="document.powform.pownr.value =
    Math.pow(document.powform.nr1.value,
    document.powform.nr2.value)"> =
  <INPUT name="pownr" type="text" size="5"
    value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

În figura 10 se poate vedea o posibilă imagine a programului de navigare, după încărcarea documentului HTML anterior.

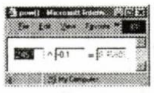


Fig. 10 Exemplu de utilizare a funcției `pow()`

Funcția random()

Pentru a genera numere aleatoare, limbajul *JavaScript* pune la dispoziție funcția `random()`. Această funcție va genera un număr pseudoaleator cuprins în intervalul `[0,1)`.

Funcția round()

Funcția `round()` returnează numărul întreg cel mai apropiat de un număr x transmis ca parametru. În cazul în care partea fracționară a numărului x este mai mică decât `0.5` funcția `round()` returnează aceeași valoare ca și funcția `floor()`, iar dacă partea fracționară a numărului x este mai mare sau egală cu `0.5` funcția `round()` returnează aceeași valoare ca și funcția `ceil()`. Următorul document HTML reprezintă un exemplu de utilizare a acestei funcții:

```
<HTML>
<HEAD> <TITLE> round() </TITLE> </HEAD>
<BODY>
  <FORM name="roundform">
  The closest integer to
  <INPUT type="text" name="nr" size="5"
  value="" onchange=
  "document.roundform.roundnr.value=
  Math.round(document.roundform.nr.value)"> is
  <INPUT name="roundnr" type="text" size="5"
  value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

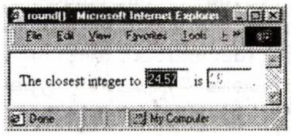


Fig. 11 Exemplu de utilizare a funcției `round()`

Funcția sqrt()

Pentru extragerea rădăcinii pătrate, limbajul *JavaScript* pune la dispoziție funcția `sqrt()`. Această funcție are ca parametru un număr pozitiv x și returnează valoarea rădăcinii pătrate a acelui număr. Evident, apelul `Math.sqrt(x)`, este echivalent cu apelul `Math.pow(x,0.5)`.

Funcții trigonometrice

Pentru operații cu funcții trigonometrice, limbajul *JavaScript* pune la dispoziție șase metode și anume `sin()`, `cos()`, `tan()`, `asin()`, `acos()` și `atan()` care pot fi folosite pentru calcul sinusului, cosinusului, tangentei, arcsinusului, arccosinusului, respectiv arctangentei unui unghi. Mărimea unghiului este transmisă ca parametru și este exprimată în radiani. Datorită faptului că în multe situații unghiurile sunt exprimate în grade, va trebui folosită formula de transformare a mărimilor din grade în radiani:

$$n^\circ = \frac{n \cdot \pi}{180} rad$$

Pentru valoarea π există o constantă predefinită *JavaScript* și anume `Math.PI`.

Proprietățile obiectului Math

Alături de proprietatea `PI` amintită anterior, obiectul `Math` mai are câteva proprietăți care desemnează valori des întâlnite în cadrul matematicii. O prezentare a acestor constante poate fi observată în tabelul 1.

Tabelul 1

Denumire	Descriere	Valoare	Valoare aproximativă
Math.E	Constanta lui Euler	e	2.7182818285
Math.LN2	Logaritm natural din 2	$ln\ 2$	0.6931471805
Math.LN10	Logaritm natural din 10	$ln\ 10$	3.3025850929
Math.LOG2E	Logaritm în baza 2 din e	$log_2 e$	1.4426950408
Math.LOG10E	Logaritm zecimal din e	$lg\ e$	0.4342944819
Math.PI	Raportul dintre circumferința unui cerc și diametrul său	π	3.1415926535
Math.SQRT1_2	Rădăcină pătrată din 1/2	$\sqrt{1/2}$	0.7071067811
Math.SQRT2	Rădăcină pătrată din 2	$\sqrt{2}$	1.4142135623

Operații cu șiruri de caractere

În *JavaScript* șirurile de caractere sunt instanțe ale unei clase speciale numite `String`. Metodele obiectelor din această clasă pot fi folosite pentru afișarea rezultatelor diferitelor calcule efectuate cu ajutorul metodelor obiectului `Math` și nu numai.

Proprietatea length

Singura proprietate a obiectelor din clasa `String` este `length`; valoarea acestei proprietăți indică numărul de caractere pe care îl conține șirul. Pentru a obține această valoare sintaxa folosită este:

```
denumire_string.Length
```

Următorul document HTML ilustrează modul de utilizare a acestei proprietăți:

```
<HTML>
<HEAD>
  <TITLE> length </TITLE>
</HEAD>
<BODY>
  <FORM name="lengthform">
  The number of characters in
  <INPUT type="text" name="string" size="5"
  value="" onchange=
  "document.lengthform.lengthnr.value=
  ocument.lengthform.string.value.length"> is
  <INPUT name="lengthnr" type="text" size="5"
  value="" disabled>.
  </FORM>
</BODY>
</HTML>
```

În figura 12 este prezentată o captură a ferestrei programului de navigare *Internet Explorer 5.0*, după încărcarea paginii Web corespunzătoare documentului HTML anterior.

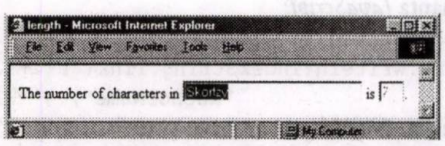


Fig. 12 Exemplu de utilizare a proprietății `length`

Funcții pentru setarea caracteristicilor de afișare

Pentru a stabili modul în care va apărea textul în cadrul paginii Web, limbajul *JavaScript* pune la dispoziție funcțiile `blink()`, `bold()`, `fontcolor()`, `italics()` și `strike()`.

Funcția blink()

Folosirea funcției `blink()` determină afișarea intermitentă a șirului de caractere în cazul în care acesta va apărea în pagina Web. Intervalul de timp dintre două apariții consecutive a textului este de o secundă. Pentru a beneficia de facilitățile oferite de această funcție, secvența *JavaScript* corespunzătoare ar putea arăta astfel:

```
function blinky() {
  var blinkString="blinks";
  document.write("A portion of this sentence"
  +blinkString.blink());
}
```

Funcția bold()

Funcția `bold()` permite afișarea textelor folosind caractere îngroșate, ea fiind echivalentă cu marcajul HTML `<B>`. Secvența *JavaScript* care trebuie folosită pentru a beneficia de această facilitate ar putea fi:

```
var boldString="is Bold";
document.write("A portion of this sentence"+
boldString.bold());
```

Funcția fontcolor()

Cu ajutorul funcției `fontcolor()` puteți stabili culoarea folosită pentru afișarea textului respectiv. Această funcție este echivalentă cu atributul **color** al marcajului `<FONT>`. Secvența *JavaScript* corespunzătoare ar putea fi:

```
var greenString = "green";
document.write("A portion of this sentence is"+
greenString.fontcolor("green"));
```

Funcția italics()

Pentru a afișa litere înclinate puteți folosi facilitățile oferite de funcția `italics()`, funcție echivalentă cu marcajul HTML `<I>`. O posibilă secvență HTML care poate fi folosită pentru afișarea caracterelor înclinate este:

```
var italicString = "is Italicized";
document.write("A portion of this sentence " +
italicString.italics());
```

Funcția strike()

Pentru a simula corecturi afișate în pagini Web puteți folosi funcția `strike()`, echivalentă cu marcajul HTML **<STRIKE>**. Aceasta poate fi folosită într-o secvență de tipul:

```
var wrong="Budapest";
var correct="Bucharest";
document.write("The capital of Romania is
"+wrong.strike()+".<BR>");
document.write("The capital of Romania is
"+correct.bold()+".<BR>");
```

Funcții pentru schimbarea tipului literelor

Pentru ca toate literele cuprinse într-un șir să devină litere mici sau litere mari, pot fi folosite funcțiile *JavaScript* `toLowerCase()` și `toUpperCase()`.

Funcția toLowerCase()

Pentru a transforma toate literele mari dintr-un șir în litere mici, funcția folosită este `toLowerCase()`. Un exemplu de utilizare este:

```
var string="Lower Case";
document.write(string.toLowerCase());
```

Funcția toUpperCase()

Pentru a transforma toate literele mici dintr-un șir în litere mari, funcția folosită este `toUpperCase()`. Un exemplu de utilizare este:

```
var string="Upper Case";
document.write(string.toUpperCase());
```

Funcții pentru manipularea șirurilor de caractere

Pentru manipularea șirurilor de caractere, limbajul *JavaScript* oferă patru funcții și anume `charAt()`, `indexOf()`, `lastIndexOf()` și `substring()`.

Funcția charAt()

Funcția `charAt()` returnează un caracter din șirul specificat. Sintaxa acestei funcții este `string.charAt(i)`, ea returnând al `i+1`-lea caracter din șirul `string`. Valoarea parametrului `i` poate fi un număr întreg din intervalul `[0,string.length()-1]`.

Funcția substring()

Funcția `substring()` returnează un subșir al șirului specificat, sintaxa ei fiind `string.substring(i,j)`. Funcția returnează un șir care conține caractere din șirul inițial începând cu poziția `i+1` și terminând cu poziția `j`. Valorile parametrilor `i` și `j` pot fi numere întregi din intervalul `[0,string.length()]`. Dacă valorile `i` și `j` sunt egale, șirul returnat va fi `"NULL"`, iar dacă `i>j` atunci apelul este echivalent cu `string.substring(j,i)`. Deși acest lucru nu este recomandat, valoarea `j` poate fi mai mare decât lungimea șirului, caz în care apelul va fi echivalent cu `string.substring(i,string.length)`.

Funcția indexOf()

Pentru a afla poziția primei apariții a unui anumit subșir într-un șir poate fi folosită funcția `indexOf()`. Sintaxa acestei funcții este `string.indexOf(s,[i])`, unde `s` este subșirul a cărui apariție este căutată, iar `i` este un parametru opțional care indică indicele începând de la care începe căutarea. Valoarea implicită a acestui parametru este 0, iar dacă este specificată o altă valoare, aceasta trebuie să fie un număr întreg din intervalul `[0,string.length()-1]`. Funcția returnează poziția în șirul `string` a primului caracter din subșirul `s`, relativ la începutul șirului, chiar dacă este prezent și parametrul `i`. În cazul în care nu este găsită nici o apariție a subșirului `s`, funcția va returna `-1`.

Funcția lastIndexOf()

Pentru a afla poziția ultimei apariții a unui anumit subșir într-un șir, poate fi folosită funcția `lastIndexOf()`. Sintaxa acestei funcții este `string.lastIndexOf(s,[i])`, semnificația parametrilor fiind aceeași ca și în cazul funcției `indexOf()`.

Funcții pentru definirea ancorelor

Pentru a avea acces la proprietățile ancorelor și legăturilor dintr-o pagină Web, limbajul *JavaScript* pune la dispoziție funcțiile `anchor()` și `link()`.

Funcția anchor()

Pentru ca un anumit text să fie considerat o ancoră poate fi folosită funcția `anchor()` care va avea ca parametru denumirea ancorei respective. De exemplu, secvențele *JavaScript* și HTML de mai jos sunt echivalente:

Varianta *JavaScript:*

```
var indexString = "Index";
document.writeln(indexString.anchor
                              ("anchorName"));
```

Varianta HTML:

```
<A name="anchorName">Index</A>
```

Funcția link()

Funcția complementară pentru `anchor()` este `link()`; această funcție va avea ca parametru denumirea țintei spre care se va face legătura. În continuare vă prezentăm două secvențe echivalente (*JavaScript* și HTML) cu ajutorul cărora poate fi generată o legătură hipertext.

Varianta *JavaScript:*

```
var indexString = "Index"
document.writeln(indexString.link("#" +
                        "anchorName") + "<BR>")
```

Varianta *HTML:*

```
<A href="#"+"anchorName">Index</A>
```

Funcții pentru poziționare

Pentru afișarea anumitor caractere ca indici superiori sau inferiori limbajul *JavaScript* pune la dispoziție funcțiile pereche `sup()` și `sub()`.

Funcția sup()

Pentru ca un anumit text să fie afișat ca indice superior va trebui să folosiți funcția `sup()`, echivalentă cu marcajul HTML <SUP>; iată un exemplu:

```
var supString="2";
document.writeln("x"+supString.sup());
```

În fereastra programului de navigare va apărea textul x^2.

Funcția sub()

Pentru ca un anumit text să fie afișat ca indice inferior va trebui să folosiți funcția `sub()`, echivalentă cu marcajul HTML <SUB>; iată un exemplu:

```
var subString="2";
document.writeln("H"+subString.sub()+"O");
```

În fereastra programului de navigare va apărea textul H_2O.

Funcții pentru dimensionare

Pentru a modifica dimensiunile textului limbajul *JavaScript* pune la dispoziție funcțiile `big()`, `fixed()`, `fontsize()` și `small()`.

Funcția big()

Pentru ca textul să fie afișat cu litere mai mari poate fi folosită funcția `big()`, echivalentă cu marcajul HTML <BIG>; modul de utilizare a acestei funcții este următorul:

```
var bigString="A BIG Font";
document.writeln(bigString.big());
```

Funcția small()

Funcția pereche a funcției `big()` este `small()`, funcție folosită pentru afișarea textului cu litere mai mici. Marcajul HTML echivalent este <SMALL>, iar un exemplu de utilizare este prezentat în continuare:

```
var smallString="A small FONT";
document.writeln(smallString.small());
```

Funcția fixed()

Pentru ca textul afișat să fie monospațiat va trebui să folosiți funcția `fixed()`, care este echivalentă cu marcajul HTML <TT>. Următoarea secvență ilustrează modul de folosire a acestei funcții:

```
var fixedSize = "This text is fixed in size.";
document.writeln(fixedSize.fixed()+"<BR>");
```

Funcția fontsize()

Pentru a avea un control mai exact asupra dimensiunii textului, poate fi folosită și funcția `fontsize()` care este echivalentă cu atributul **size** al marcajului HTML <FONT>. Această funcție are un parametru cu ajutorul căruia poate fi indicată noua dimensiune a textului, valorile admise fiind aceleași ca și în cazul atributului **size** al marcajului <FONT> și anume numere întregi între 1 și 7 pentru setarea unei noi dimensiuni, respectiv numere precedate de semnele '+' sau '-' pentru redimensionări relative la fosta dimensiune. Următorul exemplu ilustrează modul de folosire a acestei funcții:

```
<HTML>
<HEAD> <TITLE> fontsize() </TITLE>
<SCRIPT language="JavaScript">
  var showFontSize="GInfo";
  document.writeln(showFontSize.fontsize(1) +
                                    "<BR>");
  document.writeln(showFontSize.fontsize(2) +
                                    "<BR>");
  document.writeln(showFontSize.fontsize(3) +
                                    "<BR>");
  document.writeln(showFontSize.fontsize(4) +
                                    "<BR>");
  document.writeln(showFontSize.fontsize(5) +
                                    "<BR>");
  document.writeln(showFontSize.fontsize(6) +
                                    "<BR>");
  document.writeln(showFontSize.fontsize(7) +
                                    "<BR>");
   document.writeln(showFontSize.fontsize(-2)
                                + "<HR>");
  </SCRIPT>
</HEAD>
</HTML>
```

În figura 13 poate fi observată fereastra programului de navigare după încărcarea documentului HTML anterior.

Fig. 13 Exemplu de utilizare a funcției `fontsize()`

Combinarea facilităților oferite de metodele clasei String

Limbajul *JavaScript* permite și combinarea facilităților oferite de către funcțiile obiectelor instanță ale clasei `String`. Datorită faptului că, în urma apelării unei astfel de funcții, este returnat tot un obiect instanță al clasei `String`, pentru acest obiect poate fi apelată o altă metodă a acestei clase. De aceea pentru a specifica mai multe caracteristici ale unui text, trebuie să apară identificatorul obiectului instanță al clasei `String` urmat de apelurile funcțiilor corespunzătoare precedate de câte un punct. Următorul exemplu ilustrează câteva dintre combinațiile posibile.

```
<HTML>
<HEAD> <TITLE> Combining Methods </TITLE>
<SCRIPT language="JavaScript">
  function MakeArray(size){
  this.length = size;
  for(i = 1; i <= size; i++)
    this[i]='';
  return this;
  }
  colors=new MakeArray(7);
  colors[0]="#FF0000";
  colors[1]="#00FF00";
  colors[2]="#777700";
  colors[3]="#0000FF";
  colors[4]="#FF00FF";
  colors[5]="#00FFFF";
  colors[6]="#777777";

  function crazyColors(str){
          // Take each character in str and
                          // change its color
  newString="";
  for(i=0;i<str.length;i++)
  newString+=str.charAt(i).fontcolor
                                  (colors[i%7]);
  return newString;
  }

  function crazySizes(str){
          // Take each character in str and
                           // change its size
  newString="";
  for(i=0;i<str.length;i++)
    newString+=str.charAt(i).
                  fontsize((i%7)+1);
  return newString;
  }

  function crazyEmphasis(str){
          // Take each character in str and
               // change its size and color
  newString="";
  for(i=0;i<str.length;i++){
    var ch=str.charAt(i);
    var which=i%7;
  if(which==0) newString+=ch.big();
  else if(which==1) newString+=ch.small();
   else if(which==2) newString+=ch.italics();
    else if(which==3) newString+=ch.bold();
     else if(which==4) newString+=ch.strike();
      else if(which==5) newString+=ch.sub();
       else newString+=ch.sup();
  }
  return newString;
  }

  function crazyEverything(str){
          // Take each character in str and
               // change its size and color
  newString="";
  for(i=0;i<str.length;i++){
    var ch=str.charAt(i);
    var which=i%7;
    if(which==0)
     newString+=ch.big().fontsize((i%7)+1).
        fontcolor(colors[i%7]);
    else if(which==1) newString+=ch.small().
    fontsize((i%7)+1).fontcolor(colors[i%7]);
    else if(which==2)newString+=ch.italics().
    fontsize((i%7)+1).fontcolor(colors[i%7]);
     else if(which==3)newString+=ch.bold().
    fontsize((i%7)+1).fontcolor(colors[i%7]);
      else if(which==4)
                          newString+=ch.strike().
    fontsize((i%7)+1).fontcolor(colors[i%7]);
      else if(which==5)newString+=ch.sub().
    fontsize((i%7)+1).fontcolor(colors[i%7]);
      else newString+=ch.sup().
    fontsize((i%7)+1).fontcolor(colors[i%7]);
  }
  return newString;
  }
  </SCRIPT>
</HEAD>
<BODY>
  <SCRIPT language="JavaScript">
    var dString="JavaScript is fun to use!
    Just look what you can do!";
    document.write(crazyColors(dString)+
                                    "<BR>");
    document.write(crazySizes(dString)+
                                    "<BR>");
    document.write(crazyEmphasis(dString)+
                                    "<BR>");
    document.write(crazyEverything(dString)+
                                    "<BR>");

  </SCRIPT>
</BODY>
</HTML>
```

Dacă se încarcă documentul HTML anterior, fereastra programului de navigare ar putea arăta ca în figura 14.

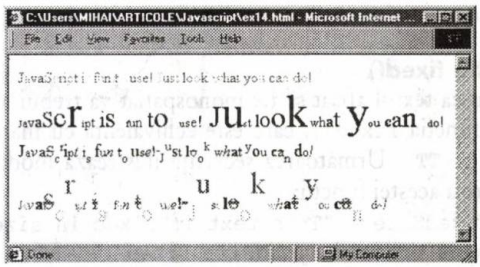


Fig. 14 Exemplu de combinare a metodelor

Mihai Scorțaru este student în anul II la Universitatea Tehnică Cluj. Poate fi contactat prin e-mail la smihai@asterix.obs.utcluj.ro.

I. CONCURSURI

Sinaia '99

În perioada 20-25 septembrie a avut loc la Sinaia a doua tabără de pregătire și selecție a Lotului Național de Informatică organizată în premieră fără sprijinul Ministerul Educației Naționale, ea fiind posibilă numai datorită sponsorilor: Computer Publishing Center București și Fundația pentru o Societate Deschisă.
Organizatorul principal a fost domnul profesor Emil Onea, inspector de informaică la Inspectoratul Județean Vrancea; alături de dânsul au contribuit la buna desfășurare a concursului profesorii Dana Lica, Clara Ionescu, Doru Popescu Anastasiu, Radu Lupșa precum și studenții Valentin Gheorghiță, Cristian Cadar, și Mihai Scorțaru. Un merit deosebit în organizarea întregii tabere l-a avut domnul Virgil Harabagiu căruia redacția GInfo, împreună cu membrii lotului național, îi adresează mulțumiri și pe această cale.
Au fost trei zile de concurs, în fiecare zi concurenții trebuind să rezolve câte trei probleme pe parcursul a cinci ore, iar în urma selecției au fost aleși cei patru reprezentanți ai României la Olimpiada Internațională de Informatică.

IOI '99

În perioada 9-16 octombrie s-a desfășurat la Antalya, în Turcia, cea de-a unsprezecea ediție a Olimpiadei Internaționale de Informatică (IOI), concurs la care au participat câte patru concurenți din 65 de țări. Părerea majorității participanților a fost că acesta a fost cel mai bine organizat IOI din istorie.
În cele două zile de concurs participanții au trebuit să rezolve câte trei probleme, ei având la dispoziție cinci ore. Țara noastră a fost reprezentată de Adrian Cârcu (Bistrița), Bogdan Dumitru (Ploiești), Ștefan Radu (Brașov) și Mihai Pătrașcu (Craiova).
La acest concurs România a obținut o medalie de aur prin Bogdan Dumitru, una de argint prin Mihai Pătrașcu și două de bronz prin Adrian Cârcu și Radu Ștefan, iar în clasamentul neoficial pe națiuni România s-a clasat a șasea.
A fost primul concurs internațional important la care evaluarea (evident automată) a durat mai puțin de două ore, iar contestațiile au fost nesemnificative. Din păcate problemele au fost deosebit de dificile, cel mai ridicat "scor" fiind de 480 de puncte din 600 posibile.

ACM 2000

În data de 23 octombrie 1999 a avut loc la Universitatea Politehnică din București etapa regională a concursului ACM în urma căreia au fost selectate echipele universităților din Europa de Sud-Est care vor participa la etapa finală care va avea loc în perioada 15-19 Martie 2000 în Orlando, Florida.
La această etapă au participat 39 de echipe, reprezentând universități din 8 țări sud-est europene: Bulgaria, Cipru, Grecia, Iugoslavia, Macedonia, Moldova, România, Turcia și Ucraina. Concurenții au avut la dispoziție cinci ore pentru a rezolva cele opt probleme propuse de o comisie internațională formată din reprezentanți ai țărilor participante.
La fel ca și anul trecut, concursul a fost câștigat de echipa Universității din București, membrii acesteia reușind să rezolve cinci din cele opt probleme.

II. SOFTWARE

Windows Millennium

Compania Microsoft a anunţat la sfârşitul lunii septembrie versiunea pentru anul viitor a sistemului de operare Windows '98 Second Edition, deocamdată în faza beta 1, cunoscută sub numele de "Millennium". Se pare totuşi că Microsoft a scos anunţul de pe site-ul său, probabil din cauza unor erori serioase descoperite foarte recent. Cu noua versiune Millennium, Microsoft va încerca să uşureze accesul la Internet şi să îmbunătăţească suportul multimedia al sistemului de operare. O versiune second beta a sistemului de operare Millennium este aşteptată până la sfârşitul acestui an.

Windows 2000 întârzie

Multe întrebări continuă să se învârtă în jurul sistemului de operare al companiei Microsoft, cu speculaţii privind data lansării, într-un moment în care compania a mai eliminat una dintre facilităţile sistemului. Între timp Microsoft şi-a anuntat partenerii că versiunea finală W2K va fi lansată în 17 februarie 2000, la Moscone Center din San Francisco. Un lucru este sigur privind versiunea finală Windows 2000: aceasta nu va mai include tehnologia bazei de date în memorie, proiectată pentru a furniza un nivel superior de interacţiuni, mai rapide, între sistemul de operare şi datele stocate.

A apărut Office 2000 în limba română

Microsoft România anunţă lansarea primei suite de aplicaţii pentru PC cu interfaţa în limba română: Microsoft Office 2000 Professional Romanian. Anunţul a fost făcut de President of Microsoft Europe Michel Lacombe, la Conferinţa de Presă care a avut loc la World Trade Center Bucureşti, Sala Seoul, joi, 7 octombrie 1999, ora 14:30.

Microsoft depăşeşte previziunile fiscale

Directorul financiar executiv al companiei Microsoft a prezentat pe scurt activităţile desfăşurate în primul trimestru al anului fiscal 2000. Compania din Seattle a depăşit previziunile specialiştilor şi analiştilor financiari, înregistrând un profit de 2,19 miliarde dolari şi o cifră de vânzări de 5,38 miliarde. Aceasta indică o creştere cu 28% faţă de aceeaşi perioadă a anului trecut, când au fost înregistrate vânzări de 4,19 miliarde dolari. În trimestrul viitor, oficialii companiei au declarat că se aşteaptă la o creştere a încasărilor de cel puţin 750 milioane dolari, adică cu cel puţin 14% mai mult faţă de primul trimestru.

Inprise JBuilder 3

Pe data de 19 octombrie 1999 compania Inprise a anunţat lansarea mediului de dezvoltare JBuilder 3 Enterprise, Solaris Edition care este disponibil pentru mediile de operare Solaris şi oferă suport pentru platforma Java 2 şi pentru viitoarea platformă Java 2 Enterprise Edition.
Se estimează că preţul acestui produs va fi de 2.499$ pentru viitorii clienţi Inprise şi 2.199$ pentru actualii clienţi. Posesorii oricărui produs Inprise Client/Server sau Enterprise pot achiziţiona JBuilder 3 Enterprise pentru "numai" 1.699$.
O versiune pentru Linux va fi disponibilă la începutul anului următor.

Semnături digitale

Comitetul judiciar al Camerei Inferioare a Congresului american a aprobat un proiect de lege care încurajează comerţul electronic prin recunoaşterea semnăturilor digitale, ca având aceeaşi putere legală ca şi semnăturile scrise de mână. Proiectul a fost aprobat de comitet săptămâna trecută, şi stipulează că, pentru valabilitatea acestei semnături pe un contract sau o tranzacţie, ambele părţi trebuie să-şi fi dat anterior acordul.

Windows vs. Linux

Computer PRESS Agora a realizat pe site-ul propriu (www.agora.ro) un sondaj pentru a determina care sunt sistemele de operare folosite de vizitatorii site-ului. Potrivit acestui sondaj 51% dintre vizitatori folosesc una dintre versiunile Microsoft Windows, 43% folosesc o versiune Linux, iar restul de 6% folosesc alte sisteme de operare. Sondajul a fost realizat în perioada 21 septembrie - 4 octombrie.

Guvernul SUA înregistrează două probleme Y2K

Agenţiile de presă anunţă că Y2K a fost cauza a două probleme minore în sistemul informaţional al Departamentului de Energie şi al Fundaţiei Naţionale de Ştiinţă, vineri 1 octombrie 1999, prima zi a noului an fiscal. Ambele probleme au fost însă remediate până la sfârşitul zilei conform declaraţiilor personalului din cadrul "Consiliului pentru trecerea în anul 2000" al preşedinţiei SUA.

Borland Delphi şi CBuilder pentru Linux

Compania Inprise anunţă că în curând va lansa versiuni pentru Linux ale mediilor de dezvoltare Borland Delphi şi CBuilder, permiţând astfel dezvoltatorilor familiarizaţi cu aceste utilitare să creeze aplicaţii native Linux sau să le porteze din Windows pe Linux relativ uşor. După lansarea acestor două utilitare, cândva la mijlocul anului viitor, Inprise va realiza şi o versiune pentru Linux a mediului său de dezvoltare Java, Borland JBuilder. Pentru a studia interesul pe care îl suscită Java pe Linux, Inprise va pune gratuit la dispoziţia dezvoltatorilor o versiune preliminară a compilatorului JBuilder pentru Linux care suportă multe din specificaţiile Java 2 Enterprise Edition.

III. HARDWARE

Noi procesoare Intel

Compania Intel a lansat luni, 25 octombrie 1999, noi variante ale procesorului Pentium III având frecvențe cuprinse între 500 și 733 MHz, precum și primul procesor Pentium III pentru notebook-uri. Graba în care a fost realizată această lansare se datorează, în mare parte și competiției, cel puțin aceasta este părerea mai multor analiști ai industriei procesoarelor. Aceștia sunt de părere că AMD determină în mare măsură cât de repede livrează Intel noi echipamente, dar și prețul acestora. Diferențele noului tip de procesoare față de cele existente încep de la design, noul Pentium III folosindu-se de designul cunoscut sub numele de Coppermine. Acesta este un proces de producție care utilizează tehnologia de 0,18 microni, care permite mărirea vitezei procesoarelor și reduce consumul de energie. Oficialii Intel au declarat, cu ocazia lansării, că noile procesoare sunt cu cel puțin 25% mai rapide decât cele existente în acest moment pe piață.

Disc hard de 73 GB

IBM a anunțat lansarea unei noi familii de drivere hard, inclusiv cel de 10.000 RPM Ultrastar 72ZX. Oferind un spațiu de stocare de 73 GB, IBM afirmă că Ultrastar 72ZX este discul hard cu cea mai mare capacitate din lume. De mici dimensiuni, cât o copertă, Ultrastar poate stoca toate informațiile care se găsesc "pe un întreg nivel al Bibliotecii Publice din New York", și ar mai rămâne loc liber. Gama este completată de încă două discuri Ultrastar 36ZX și Ultrastar 36LP, ambele cu capacități de stocare de 36 GB.

AMD în pierdere

În al treilea semestru al anului 1999 firma Advanced Micro Devices a înregistrat o pierdere netă de 105.545.000 $. Chiar dacă această firmă oferă cel mai performant procesor de pe piață (AMD Athlon) lansarea procesoarelor din această serie nu a reușit să compenseze pierderile chiar dacă vânzările au crescut față de trimestrul anterior.

Overclocking pentru Athlon

Firma Kryotech oferă deja procesoare Athlon la 900 de MHz, viteză atinsă prin overclocking. Procesoarele sunt amplasate într-o carcasă răcită special, folosind o tehnologie patentată de Kryotech. Prețul unui asemenea kit (carcasă + placă de bază + procesor) este de 2200$.

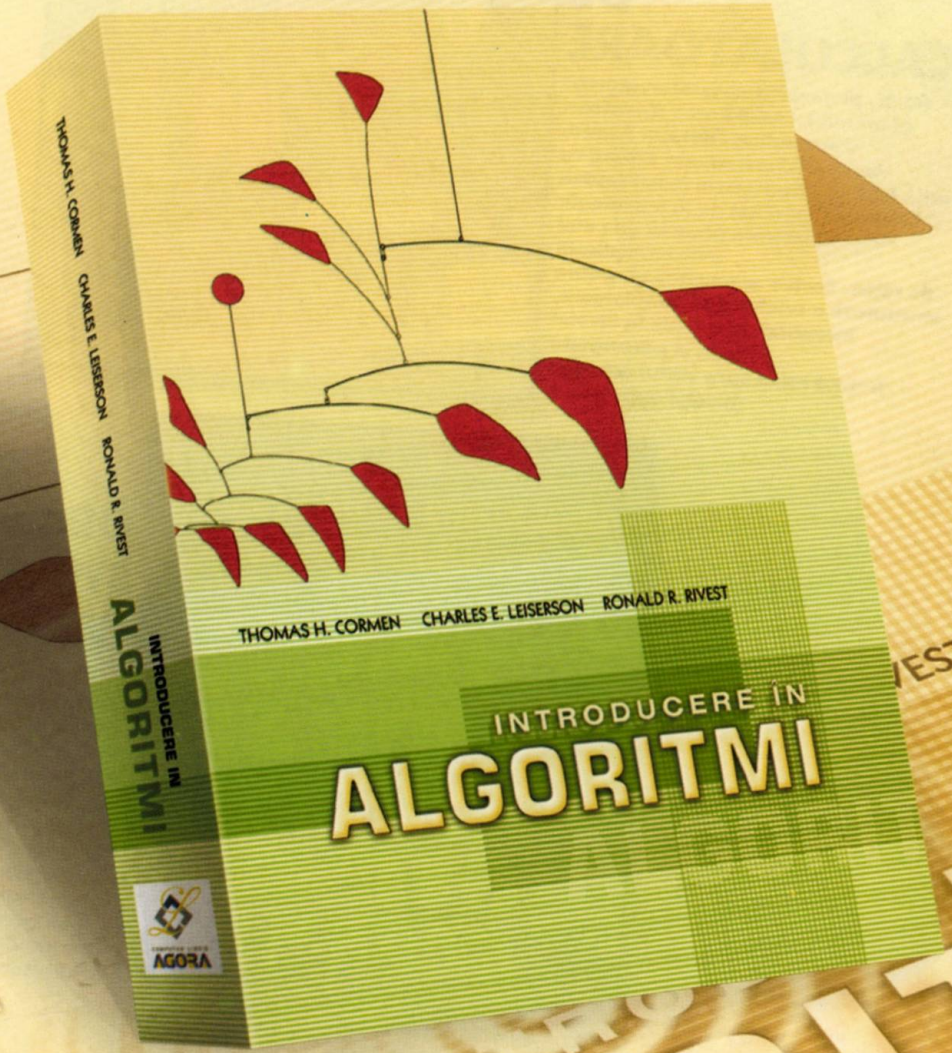
Se schimbă ordinul de mărime al vitezei procesoarelor

Se pare că, începând cu anul următor, vitezele procesoarelor se vor măsura în Gigahertzi. Firma AMD a inaugurat o nouă fabrică care va oferi produse care folosesc tehnologia de 0,18 microni, cu ajutorul căreia va fi posibilă dezvoltarea de procesoare cu o viteză de 1 GHz. Unitatea, numită Fab 30, este o parte importantă a planului companiei AMD de a rămâne în fața celor de la Intel. Primele procesoare Athlon produse în această fabrică vor funcționa la o viteză de aproximativ 900 MHz și vor fi livrate probabil la sfârșitul acestui an, iar primele procesoare Athlon cu viteze de ordinul gigahertzilor vor apărea la începutul anului următor. Oficialii Intel afirmă că și această firmă este capabilă să producă procesoare cu viteze de peste 1 GHz și că acestea vor fi disponibile pe piață la sfârșitul anului 2000 sau începutul anului 2001.

COMPUTER LIBRIS AGORA

Este în curs de apariție și în limba română celebra carte de ALGORITMI scrisă de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, publicată de MIT Press, SUA. Este o carte care de la prima ei apariție a fost retipărită, numai în Statele Unite, de 21 ori.

Această carte oferă o introducere exhaustivă a studiului modern al algoritmilor. Analiza și proiectarea algoritmilor tratați este accesibilă și asigură condițiile unui studiu, pe de o parte fundamental, pe de altă parte profund.

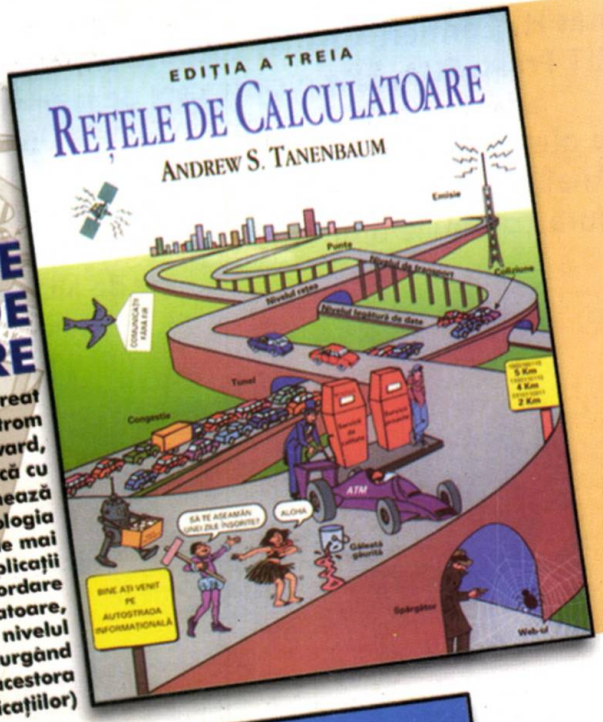


COMPUTER LIBRIS AGORA

Pentru comenzi sau informații suplimentare puteți suna la tel. 064-192422

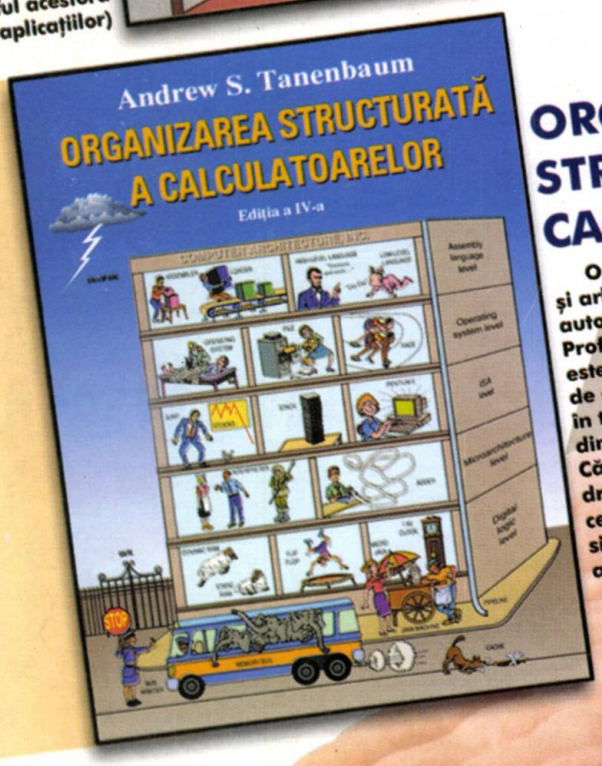
COMPUTER PRESS AGORA

ANDREW S. TANENBAUM



REȚELE DE CALCULATOARE

Autor, profesor și cercetător, laureat al premiului ACM Karl V.Karlstrom Outstanding Educator Award, Andrew S.Tanenbaum explică cu grijă felul în care rețelele funcționează în interior, pornind de la tehnologia hardware până la cele mai utilizate aplicații de rețea. Cartea urmează o abordare structurată a rețelelor de calculatoare, începând de jos, (de la nivelul fizic) și parcurgând treptat calea spre vârful acestora (nivelul aplicațiilor)



ORGANIZAREA STRUCTURATĂ A CALCULATOARELOR

O carte celebră de inițiere în structura și arhitectura calculatoarelor scrisă de un autor nu mai puțin celebru. Profesorul Andrew Tanenbaum este unul din cei mai faimoși autori de cărți de specialitate cunoscut în toate facultățile de calculatoare din lume.
Cărțile sale sunt utilizate drept cărți de referință pentru cel puțin trei domenii: rețele, sisteme de operare și arhitectura sistemelor de calcul.

COMPUTER PRESS AGORA
Str. T.Vladimirescu Nr.63/1 (CP 230/1) • Tel. 065-166516 Fax 065-166290
www.agora.ro

Gazeta de informatica
Revista viitorilor specialiști în tehnologia informației
Decembrie 1999
Vol. 9/8
Lei 15.000

din sumarul numărului:

GEOMETRIE COMPUTAȚIONALĂ

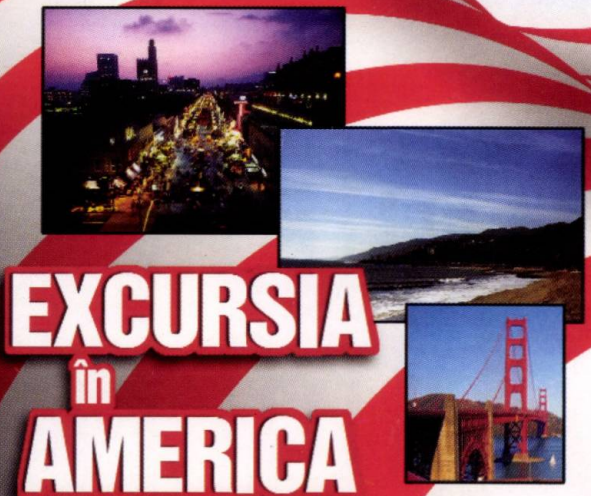
UN MODEL pentru problema orarului

CONCURSURI
Clasament Runda 1

SERIAL
Teoria grafurilor

PROBLEME PROPUSE BALCANIADA '99

Galaciuc '99 - concurs de info pentru gimnazi



EXCURSIA în AMERICA

PREMIUL LUNII

Concurs pentru elevi și studenți organizat de Gazeta de Informatică

COMPUTER LIBRIS AGORA

MANUAL Clasa a IX -a
CONFORM PROGRAMEI ȘCOLARE

Informatică

PROFIL MATEMATICĂ-INFORMATICĂ

DOINA RANCEA
GABRIELA BĂLAN
IOANA CUCU
DIANA GRUIȚĂ

AGORA

Manualul este astfel conceput încât să permită învățarea noțiunilor prezentate cu ușurință, realizarea unor aplicații utile, individual sau în grupuri de elevi. Sunt incluse multe probleme rezolvate și propuse din care profesorul poate să aleagă, în funcție de nivelul general al clasei și importanța și dificultatea problemelor. Problemele mai dificile de la sfârșitul fiecărui capitol ar putea fi date elevilor ca "miniproiecte" de aplicații în scopul dezvoltării lor cu munca în echipă. Autorii acestui manual au căutat să ofere un instrument didactic prin utilizarea căruia să crească eficiența procesului de predare și învățare și să împartă propria lor experiență didactică, acumulată pe parcursul a peste 25 de ani.

Limbaj
Algo
fundam

Clasa a X -a

MANUAL Clasa a X -a
CONFORM PROGRAMEI ȘCOLARE

Limbajul Pascal
Algoritmi fundamentali

LIMBAJE DE PROGRAMARE ȘI BAZELE INFORMATICII
APLICAȚII PRACTICE DE LABORATOR

DOINA RANCEA

AGORA

Manualul prezintă tehnici și metode de programare ca: recursivitate, alocare dinamică, metodele backtracking, programare dinamică, greedy, divide et impera și branch and bound. Prezentarea este accesibilă, iar aplicațiile sunt diverse: grafică (fractali), jocuri, teoria grafurilor. Manualul a fost conceput, plecând de la ediția 1993 (Limbajul Turbo Pascal, clasa a X-a) care a fost regândit substanțial și îmbogățit, astfel încât să fie accesibil unui elev de clasa a X-a. Ediția anterioară s-a bucurat de un succes deosebit, fiind apreciată de mii de cititori.

Două noi apariții

Pentru comenzi și informații suplimentare ne puteți contacta la tel. 064-192422

COMPUTER LIBRIS

Vol. 9, Nr. 8/1999

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Au mai colaborat:
Cristian Cadar (București)
Angel Proorocu (București)
Vlad Bazon (Vaslui)
Doru Popescu Anastasiu (Slatina)
Sorin Groza (Aiud)

Tehnoredactare:
Dan Crețu, *Bogdan Crețu, Mihai Uricaru, Călin Borz*
Corectura:
Carmen Bucur
Coperta și grafica:
Mircea Drăgoi

Adresa pentru corespondență:
C.P. 230-1, 4300 Tg-Mureș
Adresa e-mail: office@ginfo.ro
Tel: 064/192422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359
Editor: Computer Press Agora SRL
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș.
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Gabriela Bucșa (gbucsa@agora.ro)
Gabriel Piscupescu (gpiscup@agora.ro)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

sumar

probleme.. PAG. 4

propuse pentru REZOLVARE

Olimpiada Balcanică de Informatică, Ioannina, Grecia - 1999

soluții.. PAG. 7

probleme REZOLVATE

Soluțiile unor probleme propuse în cadrul taberei de pregătire de la Sinaia

concurs.. PAG. 14

Rezultatele primei runde

Clasament

Problemele rundei a treia

actual.. PAG. 51

I. Călătoria premianților

II. SOFTWARE

III. HARDWARE

IV. REVISTA presei

serial.. PAG. 23

Introducere în teoria GRAFURILOR
Cristian CADAR

Pagini WEB cu JavaScript
Mihai SCORȚARU

focus.. PAG. 34

GEOMETRIE computațională
Angel PROOROCU

Un MODEL pentru problema ORARULUI
Vlad BAZON

begin.. PAG. 43

GĂLĂCIUC '99
Doru POPESCU ANASTASIU

TOAMNA se numără...
...ALGORITMII
Sorin GROZA

Din 25 octombrie 1999
Preț Super Special pentru abonați
449 $
Calculator Alpis (fără monitor) în configurația prezentată în campania de abonamente Agora
BRINEL

BRINEL Business Partners:
Cluj-Napoca: BRINEL, Blvd. N. Titulescu #4, tel: 064-414.610
* Alba-Iulia: Info DSC SRL, 058-827.144
* Baia Mare: Multinet SRL, 062-222.201
* București: BRINEL, 094-547.804
* Brașov: Cosidor Computer, 068-310.534
* Deș: Eustel Prod Com, 064-433.713
* Râmnicu Vâlcea: Cibipro SRL, 050-734.894
* Sibiu: Alfasoft SA, Sucursala, 069-213.962
* Suceava: Globe Seattle PC, 030-215.497
* Târgu Mureș: Electro-Orizont SRL, 065-168.807
* Timișoara: AS Computer

The most productive C++ tool for Windows® 2000 and the Web.

Put power and productivity to work with Visual C++.

Microsoft® Visual C++® 6.0 is an award-winning tool that helps you develop the highest-performance business solutions for the Windows® 2000 operating system and the Web.

More Productive Development

- **Maximize productivity.** Speed development time and write less code with new IntelliSense® technology—get on-the-fly programming assistance in the editor.
- **Get faster turnaround.** Edit code right in your debugging session and continue without rebuilding, allowing you to maintain your application's state.

More Productive Development

- **Ensure your Web applications are scalable and reliable.** Develop scalable, Web-based applications with new transaction support for Microsoft Transaction Server (MTS) and Internet Information Server (IIS).
- **Customize your Internet solutions.** Take advantage of new classes and wizards for easy customization of your Microsoft Internet Information Server–based applications with MFC.

Create Faster Applications

- **Achieve peak performance.** New compiler optimizations have been added to ensure that executables run more quickly. And you can compile large projects up to 30 percent faster without sacrificing any optimizations.
- **Take advantage of flexible data access.** Use OLE DB wizards to get the highest-performance access to any data source.

WIN100 WINDOWS MAGAZINE

Win100 Award

The best and most innovated product of the year, May 1999

Game front line AWARD 1998

FrontLine Award,

Game Developer Magazine, June, 1998

Datamation Product of the Year

Datamation Product of the Year Award (Visual C++ Enterprise Edition)

Datamation magazine, April, 1998.

What's new in Visual C++:

Windows 2000 Developer's Readiness Kit

Learn how to take advantage of new Windows 2000 features through in-depth, step-by-step training and technical resources —find out how to gain greater scalability, lower cost deployment and maintenance, and the confidence that your current applications are Windows 2000 ready.

- *Get a jump-start on the new Active Directory Service Interfaces (ADSI).*
- *Learn how to build distributed applications with COM+, including integrated component management services, support for transaction processing between objects, and support for dynamic load balancing of objects across servers.*
- *Take advantage of powerful distributed application services such as message queuing, object pooling, and more. Learn to create self-repairing applications with the new Visual Studio Installer that features easy installation, rollback and repair.*
- *Get an overview of the new Windows 2000 Web Services, Core Services, Microsoft Management Console, and much more.*

Microsoft Data Engine (MSDE)

Create desktop and shared solutions that easily scale up to SQL Server 7.0 using the new Microsoft Data Engine (MSDE):

- *100% compatible with SQL Server 7.0.*
- *Royalty-free distribution of your MSDE-based applications.*
- *Reliability and features of SQL Server 7.0.*

MSDN Library Subscription

Get the essential programming information you need at your fingertips for Windows 2000 and beyond with the MSDN™ Library.

- Now includes a year's worth of updates delivered to you via an MSDN Library Subscription

Visit the Microsoft® Visual C++® Web site at:
http://msdn.microsoft.com/visualc/

© 1999 Microsoft Corporation. All rights reserved. Microsoft, ActiveX, BackOffice, MSDN, Visual Basic, Visual FoxPro, Visual InterDev, Visual J++, Visual SourceSafe, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

Microsoft®

NESIGURANȚE și certitudini

Va fi un editorial ciudat. Începe prin a vă ruga să citiți mai întâi jurnalul de excursie a lui Mihai Scorțaru de pe pagina 51... L-ați citit? E important. Doar așa veți înțelege motivele care m-au determinat să scriu rândurile care urmează.

Atunci când am citit relatările despre excursia olimpicilor în Statele Unite (organizată de Ministerul Tineretului și Sportului) am simțit cum se conturează în mine un sentiment ciudat, asemănător cu revolta. De ce aș fi revoltată?

Când am aflat prima dată despre excursie, am fost nespus de încântată pentru șansa și bucuria acestor copii de a călători, de a vedea lumea. Eram și sunt recunoscătoare ministerului amintit că se gândește la o modalitate de a "răsplăti" eforturile și performanțele realizate de cei mai talentați elevi din România care au cucerit premii și medalii la olimpiade internaționale. Care este atunci problema? Recunosc, în calitate de redactor m-am obișnuit să citesc și printre rânduri; de asemenea, recunosc că sunt sensibilă la această "dilemă" la modă care îi preocupă pe mulți dintre locuitorii României - de "a pleca"... (Nu săriți - citiți mai departe, am și eu dreptul la opinie.)

Deci s-o luăm încetișor. Se duc olimpicii în "State" și organizatorii se grăbesc să-i ducă la "Universitate" și să le explice cum pot (sau cum "trebuie") să depună cerere pentru a se înscrie la universitatea respectivă. Și nu doar la una. La mai multe. Să poată alege năzdrăvanii, nu? Români "plecați" de mai mulți ani, se grăbesc să le sugereze să "piardă" avionul de întoarcere în România. Face parte din adevăr că au fost și câțiva care, mai "patrioți" sau mai realiști, poate chiar mai cinstiți, dimpotrivă - i-au îndemnat, similar domnului profesor Manolescu (într-o emisiune televizată) să încerce să facă carieră mai întâi - acasă.

Revenind la ceea ce emană din jurnalul de excursie, mă deranjează acest exhibiționism ciudat, aproape guvernamental cu care au fost duși "copiii noștri" de lângă părinții lor, profesorii lor în "țara promisiunilor" și oferiți pe tavă unei țări, de altfel admirabile, dar care de zeci de ani își refface rezerva de talente din țări ca România. De ce nu îi lăsăm pe acești tineri în pace? Dacă tot i-am dus în America, nu era suficient (cu, sau fără vizite la universități) să se plimbe, să se uite și să decidă ei dacă îi interesează sau nu, cum se pot înscrie la o universitate americană?

Sper că cel puțin de demagogie nu mă acuzați. Cred în libertate, o doresc pentru toți locuitorii Terrei, deci, evident și copiilor noștri. Lăsați-i să strângă informații, să privească această lume plină de contradicții, nu-i "momiți" înainte să se maturizeze măcar un pic, să fie în stare să-și dea seama că "palatul" din Los Angeles nu are aceeași valoare dacă nu îl pot arăta și unor români, pentru că în Los Angeles sunt mult prea multe "palate". Lăsați-i să-și dea seama că mult visatul confort este fără importanță dacă nu îl poți împărți cu cei dragi. Lăsați-i să înțeleagă că noii prieteni nicicând nu vor putea înlocui prietenii din adolescență, cei cu care vorbesc aceeași limbă, învățată de la mama. Credeți că greșesc? Înseamnă că avem alt sistem de valori, alte priorități și deci eu vorbesc degeaba. Dar încă o dată: de unde știm că nimeni nu are sensibilități și "comori" ca ale mele? De unde știm că în România nimeni nu prețuiește amintirile din copilărie, nu își iubește limba maternă și în general, nu ține la rădăcinile lui?

Parafrazând replica domnului profesor Manolescu s-ar putea spune că o carieră realizată în România este o "victorie" mai deosebită tocmai din cauza că în România este mai greu: nimeni nu te cheamă (ca în Silicon Valley) să-ți propună în schimbul unei idei, milionul de dolari. În plus, sunt destui care cu toate posibilitățile oferite de America nu reușesc să își împlinească visele nici acolo. Dar vor lupta din răsputeri să creeze imaginea falsă că da, acolo "s-au realizat". E jenant, e chiar tragic să recunoști că nu ai reușit nici în America, nu? Și uite așa, viața se transformă într-o imensă farsă, într-o singură minciună. Pe de altă parte, toată lumea știe că într-o țară cu o economie puternică șansele sunt nelimitate și mulți sunt care reușesc. Dar de aici nu rezultă că noi (Ministerul Tineretului și Sportului, noi profesorii sau părinții lor) trebuie să-i convingem și să-i împingem în afara granițelor în speranța unei case care "se confundă cu un palat". În fond totul se reduce la a-i încuraja să nu răspundă provocării esențiale a propriei vieți, acasă în țara lor. Pleacă sau rămân? Trebuie să fie decizia lor. Și asta după ce și-au clarificat sistemul de valori.

Clara Ionescu

PROBLEME propuse pentru REZOLVARE

Balcaniada de informatică a fost organizată anul acesta în orașul Ioannina, în Grecia. Vă propunem să încercați să rezolvați problemele cu care s-au confruntat concurenții la această competiție.

P089901: Drumuri pavate

Într-o zonă izolată a unei țări sărace există `N` orașe legate prin drumuri. Între oricare două orașe există drum de legătură, nu neapărat directă. Toate aceste drumuri sunt nepavate. Guvernul dorește sa paveze unele drumuri, astfel încât sa fie îndeplinite condițiile:

- există drum pavat minimal între oricare două orașe, și
- în cazul în care s-ar elimina oricare dintre drumurile pavate, s-ar pierde legătura pe drum pavat dintre unele orașe.

În scopul de a stabili cel mai convenabil mod de pavare guvernul dorește să afle numărul tuturor posibilităților de pavare.

Date de intrare

Scrieți un program care citește datele de intrare din fișierul **`INPUT.TXT`**, după cum urmează:

Prima linie conține două numere întregi pozitive, separate printr-un singur spațiu. Primul număr reprezintă numărul de orașe `N` ($2 \le N \le 20$), al doilea reprezintă numărul de drumuri directe `R` ($N-1 \le R \le 190$).

Următoarele `R` linii precizează legăturile directe dintre orașe sub forma unor perechi de numere întregi `K` și `L`, separate prin spațiu, reprezentând faptul că orașul `K` este legat de orașul `L` ($0 \le K, L \le N-1$) printr-un drum direct.

Date de ieșire

Rezultatul se va scrie în fișierul **`OUTPUT.TXT`** pe prima linie sub forma unui număr întreg, reprezentând numărul diferitelor posibilități de pavare.

Exemplu

INPUT.TXT	OUTPUT.TXT
4 4 0 2 0 1 0 3 1 3	3

P089902: Jocul Flip

Jocul *"flip"* se joacă de o singură persoană care are la dispoziție o tablă având `M` linii și `9` coloane pe care există plăcuțe colorate pe o parte cu alb și pe cealaltă cu negru. Când o plăcuță albă se întoarce, ea devine neagră, și invers.

La fiecare mutare jucătorul întoarce o întreagă linie sau coloană, schimbând culoarea plăcuțelor în mod corespunzător.

Scopul jocului este de a lăsa pe tablă un număr minim de plăcuțe negre printr-un număr oarecare de mutări.

Date de intrare

Datele de intrare se citesc din fișierul text **`INPUT.TXT`**. Prima linie conține un număr întreg pozitiv `M` ($1 \le M \le 1000$), reprezentând numărul de linii ale tablei. Următoarele `M` linii consecutive conțin exact `9` caractere de "`0`" sau "`1`", separate printr-un singur spațiu, unde "`0`" corespunde feței albe, iar "`1`" corespunde feței negre.

Date de ieșire

Fișierul **`OUTPUT.TXT`** va conține un singur număr întreg, reprezentând numărul minim de plăcuțe care vor rămâne negre pe tablă, conform unei strategii de joc optime.

Exemplu

INPUT.TXT	OUTPUT.TXT
4 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	1

P089903: De pază în galeria de arte

Galeria *'El Greco'* constă dintr-o singură încăpere ale cărei pereți pot avea două orientări: din `Nord` spre `Sud` sau din `Vest` spre `Est`. Perimetrul podelei formează o singură linie poligonală, închisă. În interiorul încăperii nu există nici

un fel de obiecte, în schimb, pe fiecare perete există picturi valoroase și proprietarul galeriei dorește să plaseze un gardian în încăpere pentru a le păzi. În consecința proprietarul dorește să afle, dacă există un punct din încăpere, de unde se pot vedea toți pereții în întregime. Un perete se consideră vizibil chiar dacă gardianul este așezat de-a lungul liniei peretelui.

Date de intrare

Datele de intrare se vor citi din fișierul **INPUT.TXT**, după cum urmează:

Pe prima linie se află un număr N ($4 \le N \le 1000$) reprezentând numărul colțurilor încăperii. Pe următoarele N linii sunt scrise perechi de numere întregi separate printr-un spațiu, X și Y reprezentând coordonatele colțurilor în sensul acelor de ceasornic ($0 \le X$, $Y \le 1000$).

Date de ieșire

Rezultatul se va scrie în fișierul **OUTPUT.TXT**. Pe singura linie a fișierului se vor scrie coordonatele unui punct din încăpere de unde gardianul poate vedea toți pereții; coordonatele vor fi separate printr-un singur spațiu. În cazul în care există mai multe soluții, în fișier se va scrie una singură.

În cazul în care nu există nici un asemenea punct, în fișier se va scrie: "NO POINT".

Exemplu

INPUT.TXT	OUTPUT.TXT
6	70 10
0 0	
0 50	
50 50	
50 100	
100 100	
100 0	

P089904: Statistici la Poliție

Un ofițer de poliție este însărcinat să realizeze o statistică cu privire la criminalitatea dintr-o metropolă. Ofițerul trebuie să determine numărul crimelor care au avut loc într-o zonă convexă din oraș.

Date de intrare

Formatul fișierului de intrare **INPUT.TXT** este următorul:

- pe prima linie se află scris numărul întreg N ($3 \le N \le 1000$) reprezentând numărul de colțuri ale zonei convexe analizate;
- pe următoarele N linii sunt scrise câte două numere întregi, aparținând intervalului [-1000, 1000] separate printr-un singur spațiu, reprezentând coordonatele colțurilor în ordinea în care se succed pe marginea zonei definite;
- următoarea linie conține un număr întreg M ($1 \le M \le 10000$) reprezentând numărul crimelor comise în metropolă;
- fiecare din următoarele M linii conține câte două numere întregi X și Y (aparținând intervalului [-1000, 1000] separate printr-un spațiu. Aceste numere reprezintă coordonatele locului în care s-a produs o crimă.

Date de ieșire

Rezultatul se va scrie în fișierul **OUTPUT.TXT** și constă dintr-un singur număr întreg. Acesta reprezintă numărul crimelor comise în interiorul sau pe frontiera zonei analizate.

Exemplu

INPUT.TXT	OUTPUT.TXT
4	1
0 0	
0 100	
100 100	
100 0	
2	
50 50	
-30 50	

P089905: Râul Highwater

Două orașe, *Westmouth* și *Eastmouth*, sunt situate pe malul vestic, respectiv malul estic al râului *Highwater*. *Westmouth* se află la sud de *Eastmouth*. Malurile râului sunt linii poligonale astfel încât o linie orientată de la est la vest intersectează fiecare mal în exact un punct.

Căpitanul Hook navighează des între cele două orașele. Din dorința de a minimiza cheltuielile pentru combustibil, el ar vrea să afle cea mai scurtă rută de la *Westmouth* la *Eastmouth*.

Date de intrare

Fișierul de intrare **INPUT.TXT** conține pe prima linie două numere întregi M și N ($2 \le M$, $N \le 2000$) reprezentând numărul colțurilor de pe malul vestic, respectiv estic.

Pe următoarele M linii se află câte două numere întregi X și Y ($0 \le X$, $Y \le 3600$). Aceste numere reprezintă coordonatele colțurilor de pe malul vestic începând de la *Westmouth* la un punct care se află pe aceeași latitudine (adică având aceeași coordonată Y) cu *Eastmouth*.

Următoarele N linii conțin câte două numere întregi X și Y ($0 \le X$, $Y \le 3600$) care reprezintă coordonatele colțurilor de pe malul estic începând cu un punct aflat la aceeași latitudine geografică (adică având aceeași coordonată Y) ca *Westmouth*, către *Eastmouth*.

Date de ieșire

Fișierul de ieșire **OUTPUT.TXT** conține linii în care sunt scrise câte două numere întregi X și Y, separate printr-un spațiu, reprezentând coordonatele colțurilor rutei între *Westmouth* și *Eastmouth*. (Este evident că prima linie trebuie să conțină coordonatele orășelului *Westmouth*, iar ultima linie conține coordonatele orășelului *Eastmouth*.)

Exemplu

INPUT.TXT	OUTPUT.TXT
3 3	0 0
0 0	50 50
50 50	50 100
0 150	100 150
100 0	
50 100	
100 150	

P089906: Cunoștințe

La o petrecere participă persoane care pot să îi cunoască pe toți ceilalți, pe unii dintre ei sau pe nici unul.

Dorim să aflăm dacă există o modalitate de a așeza toate persoanele într-o coadă, astfel încât pentru orice persoană aflată în coadă, în spatele său se află cunoștințe și alte persoane. Cunoștințele unei persoane, aflate în spatele acestuia trebuie să se cunoască între ei.

Date de intrare

Fișierul de intrare **INPUT.TXT** are următoarea structură:

- prima linie conține numărul N ($1 \le N \le 50$) de persoane participante la petrecere;
- cea de a doua linie conține numărul M de perechi de persoane care se cunosc reciproc ($0 \le M \le 1225$);
- fiecare dintre următoarele M linii conține două numere întregi I și J ($1 \le I, J \le N$) separate printr-un singur spațiu, indicând faptul că persoanele I și J se cunosc reciproc.

Date de ieșire

În fișierul de ieșire **OUTPUT.TXT** pe prima linie se va scrie "YES" în cazul în care există o modalitate de plasare a persoanelor într-o coadă, conform cerințelor problemei; în caz contrar în fișier se va scrie "NO". Dacă răspunsul este "YES", următoarea linie din fișier va conține o ordine convenabilă a celor N persoane (de la prima persoană la ultima). Aceste numere vor fi separate printr-un spațiu.

Exemplu

INPUT.TXT	OUTPUT.TXT
5	YES
7	1 2 4 5 3
1 2	
3 1	
2 3	
2 4	
4 3	
3 5	
4 5	

BOI '99

Această Balcaniadă a fost organizată de Universitatea din Ioannina (G. Akrivis, L. Palios), de Greek Computer (G. C. S. - președinte S. Bakogiannis), de Universitatea din Salonic (I. Manolopoulos) care cu ajutorul sponsorilor au reușit să asigure desfășurarea concursului și a unei excursii pe malul Mării Ionice în condiții foarte bune.

România a fost reprezentată cu glorie de echipa formată din:

Adrian Cârcu - Bistrița - 175 puncte - aur

Tudor Leu - Focșani - 151 puncte - argint

Angel Proorocu - București - 140 puncte - argint

Daniel Oprean - Cugir - 89 puncte - bronz

La această ediție a Balcaniadei au participat în total 35 de concurenți din Albania, Bulgaria, Cipru, Iugoslavia, Macedonia, România, Turcia și, evident, Grecia. S-au acordat 2 medalii de aur, 5 de argint și 9 de bronz. Din păcate, nici un concurent nu a obținut punctaj maxim (200 puncte), cel mai bun rezultat fiind al lui Adrian Cârcu.

SOLUȚIILE problemelor propuse

Vă prezentăm în continuare câteva din soluțiile oficiale pentru problemele propuse în cadrul taberei de pregătire de la Sinaia. Soluțiile aparțin autorilor problemelor

P079902: La cinema

Observăm că există trei cazuri:

- dacă permutarea dată este cea identică, numărul necesar de minute este 0;
- dacă permutarea dată conține numai cicluri de lungime 2, numărul necesar de minute este 1;
- dacă permutarea conține cel puțin un ciclu de lungime mai mare decât 2, atunci numărul necesar de minute este 2.

În cazul 3 se acționează separat pentru fiecare ciclu din interiorul permutării.

Fie ciclul de lungime p:

$a_1\ a_2\ a_3\ \ldots\ a_{p-2}\ a_{p-1}\ a_p$

$a_2\ a_3\ a_4\ \ldots\ a_{p-1}\ a_p\ a_1$

În primul minut schimbăm a_1 cu a_p, a_2 cu a_{p-1} etc. Dacă ciclul are lungime impară, atunci elementul din mijloc rămâne neschimbat. Se va ajunge la următoarea configurație:

$a_1\ a_2\ a_3\ \ldots\ a_{p-2}\ a_{p-1}\ a_p$

$a_1\ a_p\ a_{p-1}\ \ldots\ a_4\ a_3\ a_2$

În cel de-al doilea minut schimbăm a_2 cu a_p, a_3 cu a_{p-1} etc. Remarcăm că astfel am și ajuns la configurația dorită:

$a_1\ a_2\ a_3\ \ldots\ a_{p-2}\ a_{p-2}\ a_p$

$a_1\ a_2\ a_3\ \ldots\ a_{p-2}\ a_{p-1}\ a_p$

Se procedează în mod analog pentru fiecare ciclu, ajungându-se în final la permutarea identică.

Listing: CINEMA.CPP

```
#include<stdio.h>
#include<mem.h>
#include<process.h>

int n,a[1001],all2=1,v[1001],total[3],MAX=0;
FILE *f;

void citire(){
  f=fopen("CINEMA.IN","r");
  fscanf(f,"%d",&n);
  for(int i=1;i<=n;i++)
    fscanf(f,"%d",&a[i]);
  fclose(f);
}

void identica(){
  memset(v,0,sizeof(v));
  int da=1;
  for(int i=1;i<=n;i++)
    if(a[i]!=i)
      da=0;
    else v[i]=1;
  if (da){
    f=fopen("CINEMA.OUT","w");
    fprintf(f,"0");
    fclose(f);
    exit(0);
  }
}

void gaseste_ciclu(int k){
  v[k]=1;
  int i=a[k],lung=1;
  while (i!=k) v[i]=1,i=a[i],lung++;
  if (lung>2) all2=2;
  total[1]+=lung/2;
  total[2]+=(lung-1)/2;
}

void prelucreaza_ciclu(int k){
  int c[1001],i=a[k],lung=1;
  v[k]=1;
  c[lung]=k;
  while (i!=k) c[++lung]=i,v[i]=1,i=a[i];
  for(i=1;i<=lung/2;i++){
    fprintf(f,"%d %d\n",c[i],c[lung-i+1]);
```

```
    int aux=a[c[i]];
    a[c[i]]=a[c[lung-i+1]];
    a[c[lung-i+1]]=aux;
  }
  if (lung>MAX) MAX=lung;
}

void prelucrare(){
  total[1]=total[2]=0;
  for (int i=1;i<=n;i++)
    if (!v[i]) gaseste_ciclu(i);
  f=fopen("cinema.out","w");
  fprintf(f,"%d\n",all2);
  for (int pas=1;pas<=all2;pas++){
    fprintf(f,"%d\n",total[pas]);
    memset(v,0,sizeof(v));
    for (i=1;i<=n;i++)
      if (!v[i]) prelucreaza_ciclu(i);
  }
  fclose(f);
  printf("%d\n",MAX);
}

void main(){
  citire();
  identica();
  prelucrare();
}
```

P079905: Fire

Există cel puțin două soluții care îi permit electricianului să descopere corespondența dintre capetele firelor în doar două mișcări, adică o coborâre în beci și o urcare în pod. Deoarece nu se poate rezolva problema în mai puțin de două mutări pentru nici un caz, aceste soluții sunt optime.

Pentru prima soluție vom considera întâi cazul în care numărul de fire este impar. Pentru a putea prezenta mai ușor acest caz, vom considera, de exemplu, numărul de fire egal cu 11.

În pod, electricianul scurtcircuitează cinci perechi de fire (ele sunt legate prin linii punctate în figura 1), lăsând liber un fir. Apoi coboară în beci și identifică capetele de jos ale perechilor scurtcircuitate. Notează capetele de jos cum se vede în figura 1 și apoi le scurtcircuitează așa cum arată liniile punctate.

Fig. 1

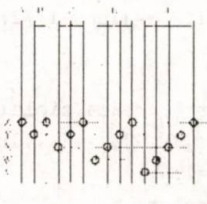


Fig. 2

Urcând din nou în pod, electricianul înlătură toate scurtcircuitele, dar lasă firele împreună pe porțiuni insulare, astfel încât perechile să fie acum identificabile. Verifică apoi continuitatea dintre firul liber (despre care știe că este capătul de sus a firului F) și un alt fir. După ce a găsit celălalt fir, el poate imediat să-l noteze cu E_2 și să-i identifice perechea E_1. Apoi verifică continuitatea dintre E_1 și un alt capăt, care, odată găsit, poate fi notat cu D_2 și perechea lui cu D_1. Continuând în același fel, capetele rămase sunt ușor identificate. Se observă că procedeul este valabil pentru orice număr impar de fire.

Pentru un număr par de fire (cu excepția lui doi) se poate aplica procedeul de mai sus puțin modificat. De exemplu, să admitem un al 12-lea fir în extrema dreaptă a figurii 1. Aceleași 5 perechi sunt scurtcircuitate, în pod rămânând două fire libere. În beci, firele sunt scurtcircuitate ca înainte, iar al 12-lea fir este notat cu G. Revenind în pod, electricianul identifică ușor pe G ca fiind singurul fir rămas liber dintre cele două fire libere inițial. Restul de 11 fire se notează așa cum s-a explicat mai sus.

Un procedeu mai eficient, care ține seama de toate cazurile, este prezentat în continuare pentru 15 fire (figura 2). Electricianul scurtcircuitează în pod firele în grupe de câte 1, 2, 3, 4 și 5 notate respectiv cu A, B, C, D, E. Ultima grupă nu trebuie să fie completă. Apoi coboară în beci și identifică grupele. Numerotează firele și le scurtcircuitează în grupele Z, Y, X, W, V. Urcând din nou în pod, electricianul înlătură scurtcircuitările. Acum el poate identifica firele fără nici o problemă. Firește, A este firul 1. Firul 3 este singurul din grupa B care este conectat cu 1. Perechea lui este 2. În grupa C, firul 6 se conectează cu 1 și firul 5 cu 2. Firul rămas în C va fi 4. Se aplică același procedeu și în continuare. Metoda se aplică identic pentru orice număr de fire.

Listing FIRE.CPP

```
#include "fire.h"

int n;

void impar(void){
  int nr,x,t,k,j,sol,i;
  int a[501],b[501],c[501],sel[501];
  for (i=1;i<=n/2;i++) Connect(i*2-1, i*2);
  GoDown();
  for(i=1;i<=n;i++) a[i]=0;
  for(i=1;i<=n;i++) b[i]=0;
  for(i=1;i<=n;i++) c[i]=0;
  sol=0;
  for(i=1;i<=n;i++)
    if(!a[i]){
      a[i]=i;
      if(!sol) sol=i;
      for(j=i+1;j<=n;j++)
        if(IsConnect(i,j)){
          if(sol==i) sol=0;
          a[j]=i;
        }
    }
```

```
  for(i=1;i<=n;i++) sel[i]=0;
  sel[sol]=1; x=sol; nr=0;
  for(j=1;j<=n;j++)
    if(!sel[j] && nr!=n/2){
      nr++;
      Connect(x,j);
      c[x]=c[j]=x;
      sel[j]=1;
      for(i=j+1;i<=n;i++)
        if(a[j]==a[i])x=i,sel[i]=1;
    }
  GoUp();
  Clear();
  sol=0;
  for(i=1;i<=n;i++)
    if(!b[i]){
      b[i]=i;
      if(!sol) sol=i;
      for(j=i+1;j<=n;j++)
        if(IsConnect(i,j)){
          if(sol==i) sol=0;
          b[j]=i;
        }
    }
  for(i=1;i<=n;i++)
    if(!c[i]) x=i;
  Solution(sol, x);
  for(i=1;i<=n/2;i++){
    for(j=1;j<=n;j++)
      if(a[x]==a[j] && x!=j) k=j;
    if(sol%2) t=sol+1;
    else t=sol-1;
    Solution(t,k);
    for(j=1;j<=n;j++)
      if(c[k]==c[j] && k!=j) x=j;
    for(j=1;j<=n;j++)
      if(b[t]==b[j] && t!=j) sol=j;
    Solution(sol,x);
  }
  Stop();
}

void par(void){
int nr,sol2,x,t,k,j,sol,i,liber,
    liber_jos,a[501],b[501],c[501],sel[501];
  for(i=1;i<=(n-2)/2;i++)
    Connect(i*2-1,i*2);
  GoDown();
  for(i=1;i<=n;i++)
    a[i]=b[i]=c[i]=0;
  sol=sol2=0;
  for(i=1;i<=n;i++)
    if(!a[i]){
      a[i]=i;
      if(!sol) sol=i;
      for(j=i+1;j<=n;j++)
        if(IsConnect(i,j)){
          if(sol==i) sol=0;
          a[j]=i;
        }
      if(sol && !sol2) sol2=sol,sol=0;
    }
  for(i=1;i<=n;i++) sel[i]=0;
  sel[sol]=sel[sol2]=1;
  liber_jos=sol2;
  x=sol; nr=0;
  for(j=1;j<=n;j++)
    if(!sel[j] && nr!=(n-2)/2){
      nr++;
      Connect(x, j);
      c[x]=c[j]=x;
      sel[j]=1;
      for(i=j+1;i<=n;i++)
        if(a[j]==a[i]) x=i,sel[i]=1;
    }
  GoUp();
  Clear();
  sol=sol2=0;
  for(i=1;i<=n;i++)
    if(!b[i]){
      b[i]=i;
      if(!sol) sol=i;
      for(j=i+1;j<=n;j++)
        if(IsConnect(i,j)){
          if(sol==i) sol=0;
          b[j]=i;
        }
        if(sol && !sol2) sol2=sol,sol=0;
    }
  if(sol>n-2) liber=sol,sol=sol2;
  else liber=sol2;
  Solution(liber,liber_jos);
  for(i=1;i<=n;i++)
    if(!c[i] && i!=liber_jos) x=i;
  Solution(sol,x);
  for(i=1;i<=(n-2)/2;i++){
    for(j=1;j<=n;j++)
      if(a[x]==a[j] && x!=j) k=j;
    if(sol%2) t=sol+1;
    else t=sol-1;
    Solution(t,k);
    for(j=1;j<=n;j++)
      if(c[k]==c[j] && k!=j) x=j;
    for(j=1;j<=n;j++)
      if(b[t]==b[j] && t!=j) sol=j;
    Solution(sol,x);
  }
  Stop();
}
```

```
void main(void){
  GoUp();
  Start();
  n=Number();
  if(n%2) impar();
  else par();
}
```

P079906: Ora de gimnastică

Numărul de configurații (o configurație reprezintă un mod de aranjare a elevilor din cerc) fiind finit (în număr de 2^n), la un moment dat configurațiile vor începe să se repete. Problema cere deci să se afle baza și perioada cu care ciclează aceste configurații. Baza reprezintă numărul de pași după care configurațiile încep să cicleze, iar perioada reprezintă lungimea unui ciclu.

Datorită numărului foarte mare de configurații, memorarea tuturor configurațiilor este imposibilă. De aceea, se va folosi următorul algoritm:

Pasul 1. Inițial, vom reține în două variabile A și B configurația inițială: NA=0, NB=0, NA și NB reprezentând câți pași am făcut pentru a realiza configurația A, respectiv configurația B, din configurația inițială.

Pasul 2. Avansăm în A cu un pas, iar în B cu 2 pași; NA=NA+1; NB=NB+1.

Pasul 3. Dacă A și B nu reținem două configurații identice, atunci repetăm Pasul 2.

Pasul 4. În acest moment NB-NA reprezintă un multiplu al perioadei. Pentru a afla efectiv perioada, pornim de la A si numărăm câți pași am realizat pentru a ajunge din nou în configurația A. Pentru a găsi baza, pornim de la configurația inițială și numărăm câți pași am realizat pentru a ajunge în configurația A.

Pasul 5. Știind baza B și perioada P, configurația la care se ajunge după K pași este configurația la care ajunge după doar B+(K-B) mod P pași.

Algoritmul de aflare a bazei și a perioadei, prezentat mai sus, se bazează pe faptul că diferența dintre NA și NB crește la fiecare pas cu o unitate, deci în mod sigur configurațiile reținute în A și B vor fi egale după un număr de pași care este egal cu lungimea perioadei sau cu un multiplu al acesteia.

Listing GIMNAST.PAS

```
{$r-,s-}
{$M 65000,0,655360}

type vector=array[-2..10001] of Char;
var init,a,b,st:vector;
    n,i,j,r:Integer;
    rule:array[0..7] of string;
    f:Text;
    nr,ntr,basis,period,npas:Longint;

procedure citire;
begin
  Assign(f,'GYM.IN'); Reset(f);
  Readln(f,ntr);
  n:=0;
  while not Seekeoln(f) do
  begin
    Inc(n);
    Read(f,st[n])
  end;
  if ntr=0
  then
  begin
    Assign(f,'GYM.OUT'); Rewrite(f);
    for i:=1 to n do Write(f,st[i]);
    Close(f);
    Halt
  end;
  for i:=1 to n do
    if st[i]='J'
    then a[i-1]:='J'
    else a[i-1]:='S';
  a[n]:=a[0];
  a[-1]:=a[n-1];
  a[-2]:=a[n-2];
  b:=a;
  init:=a;
  r:=-1;
  while not Seekeof(f) do
  begin
    Inc(r);
    Readln(f,rule[r])
  end;
  Close(f)
end;

procedure modifica(var a:vector);
var c:vector;
    i,j:Integer;
begin
  for i:=0 to n-1 do
    for j:=0 to r do
      if (a[i]=rule[j,1]) and
         (a[i-2]=rule[j,2]) and
         (a[i+1]=rule[j,3])
      then c[i]:=rule[j,4];
  a:=c;
  a[-2]:=c[n-2]; a[-1]:=c[n-1]; a[n]:=a[0]
end;

procedure transf;
begin
  Inc(npas);
  modifica(a); modifica(b); modifica(b)
end;
```

```
function egal(var a,b:vector):Boolean;
var i:Integer;
begin
  egal:=true;
  for i:=0 to n-1 do
    if a[i]<>b[i]
    then
    begin
      egal:=false;
      Exit
    end
end;

procedure baza_perioada;
begin
  a:=init;
  basis:=0;
  while not egal(a,b) do
  begin Inc(basis); modifica(a) end;
  period:=0;
  repeat
    Inc(period);
    modifica(a);
  until egal(a,b);
  if basis=period then basis:=0
end;

procedure pozitie;
begin
  if ntr<=basis
  then nr:=ntr
  else nr:=basis+(ntr-basis) mod period
end;

procedure prelucrare;
begin
  npas:=0;
  repeat transf until egal(a,b);
  baza_perioada;
  pozitie
end;

procedure afisare;
var i:Longint;
begin
  a:=init;
  if nr=0 then nr:=basis+period;
  for i:=1 to nr do modifica(a);
  Assign(f,'GYM.OUT'); Rewrite(f);
  for i:=0 to n-1 do Write(f,a[i]);
  Close(f)
end;

Begin
  citire; prelucrare; afisare
End.
```

P079907: Origami

O abordare *brute force* a acestei probleme ar cere menținerea unei structuri de date ce descrie starea colii de hârtie de pe masa mașinii. Desigur, zonele de o anumită grosime formează o reuniune de poligoane, prin urmare putem menține o listă de perechi de forma (*grosime*, *listă de poligoane*) unde *listă de poligoane* reprezintă lista zonelor colii în care avem *grosime* pliuri suprapuse.

Inițial avem deci o singură zonă (un singur poligon) de grosime 1, și anume pătratul (0, 0, 100, 100). La fiecare îndoire trebuie să facem următoarele operații:

- determinăm poligoanele "tăiate" de linia de îndoire și le descompunem în două;
- oglindim poligoanele dintr-unul din semiplane în celălalt;
- intersectăm poligoanele astfel suprapuse;
- pentru fiecare poligon rezultat din intersecție, calculăm grosimea (numărul de pliuri) ca sumă a grosimilor poligoanelor din care provine.

Desigur că o astfel de rezolvare este corectă, dar implementarea este foarte anevoioasă.

Soluția mai simplă pornește de la observația că avem nevoie doar de grosimea origami-ului în punctele candidate pentru efectuarea perforației. Cum putem calcula grosimea într-un astfel de punct? Ne uităm la ultima îndoire efectuată: dacă punctul se află în semiplanul dinspre care se face îndoirea, atunci grosimea este evident 0 (tocmai am pliat ce era acolo); dacă punctul se află în semiplanul spre care se face îndoirea, atunci pliurile din acel punct sunt cele dinainte de îndoire plus cele ce au venit în urma îndoirii. Ca urmare, grosimea este grosimea inițială în acel punct plus grosimea inițială în punctul simetric față de linia de îndoire. Bineînțeles, inițial coala are grosimea 1 în interiorul pătratului (0, 0, 100, 100) și 0 în rest.

Prin urmare, algoritmul este următorul:

```
var t:array of linie;      { lista de îndoiri }
    n:Integer;             { numărul de îndoiri }
function grosime(p:punct; k:Integer);
        { calculează grosimea în punctul p, }
                  { după primele k îndoiri }
begin
  if k=0
  then
    if p in patrat(0,0,100,100)
    then grosime:=1
    else grosime:=0
  else
    if stanga(p,t[k])
    then grosime:=grosime(p,k-1)+
           grosime(simetric(p,t[k]),k-1)
    else grosime:=0
end;
```

Implementarea recursivă are avantajul simplității, fiind o traducere directă a celor prezentate anterior. Nu insis-

tăm aici asupra calculului simetricului unui punct față de o dreaptă sau asupra determinării de ce parte se găsește un punct față de o dreaptă. Mai remarcăm faptul că numărul de apeluri ale funcției *grosime* crește exponențial cu numărul de îndoiri, dar că și în cazul primei abordări (prin crearea unei liste de zone poligonale cu grosimile corespunzătoare) complexitatea algoritmului este exponențială în numărul de îndoiri, deoarece însuși numărul de zone poligonale poate crește exponențial.

Listing ORIGAMI.PAS

```
type punct=record x,y:Real end;
const fisint='ORIGAMI.IN';
      fisout='ORIGAMI.OUT';
      eps=0.2;
var linii:array[1..10,1..2] of punct;
                              { linii de pliere }
    nrlinii,nrpuncte:Integer;
  { numar de puncte si numar de linii date }
    f,g:Text;
    i:Integer;
    b:punct;

function stanga(b,p1,p2:punct):Boolean;
var det:Real;
begin
  det:=p1.x*(p2.y-b.y)+b.x*(p1.y-p2.y)
                        +p2.x*(b.y-p1.y);
  stanga:=det>0
end;

procedure desface(var b:punct; p1,p2:punct);
  { il transforma pe b be baza simetriei }
var intersect,rezultat:punct;
    t:Real;
begin
  t:=-((p2.x-p1.x)*(p1.x-b.x)+(p2.y-p1.y)*
 (p1.y-b.y))/(Sqr(p2.x-p1.x)+Sqr(p2.y-p1.y));
  intersect.x:=p1.x+(p2.x-p1.x)*t;
  intersect.y:=p1.y+(p2.y-p1.y)*t;
  rezultat.x:=2*intersect.x-b.x;
  rezultat.y:=2*intersect.y-b.y;
  b:=rezultat
end;

function calcul(b:punct; n:Integer):Integer;
var suma:Integer;
begin
  if n=0
  then
  begin { hartia nu a fost pliata, verificam }
        { daca punctul se afla in patrat    }
    if (b.x>0) and (b.x<100) and (b.y>0)
                             and (b.y<100)
    then calcul:=1 else calcul:=0
  end
  else
  begin
   { hartia a fost pliata cel putin o data }
    if stanga(b,linii[n,1],linii[n,2])
    then
    begin
      suma:=calcul(b,n-1);
      desface(b,linii[n,1],linii[n,2]);
      suma:=suma+calcul(b,n-1);
      calcul:=suma
    end
    else calcul:=0  { punctul b se afla in }
      { partea dreapta a liniei de pliere }
  end
end;

procedure init;
var i,j:integer;
begin
  Assign(f,fisint); Reset(f);
  Assign(g,fisout); Rewrite(g);
  Readln(f,nrlinii);
  for i:=1 to nrlinii do
  begin
    for j:=1 to 2 do
      Read(f,linii[i,j].x,linii[i,j].y)
  end;
  Readln(f,nrpuncte)
end;

Begin
  init;
  for i:=1 to nrpuncte do
  begin
    Readln(f,b.x,b.y);
    Writeln(g,calcul(b,nrlinii))
  end;
  Close(f); Close(g)
End.
```

P079908: Permanent

Este evident că, similar ca și în cazul determinantului, înmulțind elementele de pe o linie (sau coloană) a matricei cu o constantă k, valoarea permanentului crește de k ori. Rezultă imediat că putem "scoate în fața permanentului" o constantă dacă împărțim o linie (coloană) la acea constantă.

Inițial trebuie să calculăm permanentul matricei

$$A=\begin{pmatrix} 1\cdot 1 & 1\cdot 2 & \cdots & 1\cdot n \\ 2\cdot 1 & 2\cdot 2 & \cdots & 2\cdot n \\ \vdots & \vdots & \ddots & \vdots \\ n\cdot 1 & n\cdot 2 & \cdots & n\cdot n \end{pmatrix}.$$

Se observă că pe linia i putem da factor comun valoarea i și permanentul matricei A devine perm A=n!•perm B, unde

$$B=\begin{pmatrix}1 & 2 & \cdots & n\\ 1 & 2 & \cdots & n\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 2 & \cdots & n\end{pmatrix}.$$

Din nou se observă că pe coloana `i` putem da factor comun valoarea `i` și permanentul matricei `A` devine `perm A=(n!)*perm C`, unde

$$C=\begin{pmatrix}1 & 1 & \cdots & 1\\ 1 & 1 & \cdots & 1\\ \vdots & \vdots & \ddots & \vdots\\ 1 & 1 & \cdots & 1\end{pmatrix}.$$

Este evident că permanentul acestei matrice este `n!` deoarece fiecare produs din dezvoltarea permanentului este 1 și există `n!` astfel de produse care sunt însumate. În concluzie avem `perm A=(n!)^3`, deci problema se reduce la a determina numărul de cifre al lui `(n!)^3`.

Orice număr zecimal `N` poate fi scris sub forma $N=10^{\lg N}$; de aici rezultă imediat că numărul de cifre al lui `N` este `[lgN]+1`. Am redus problema la calculul valoarii $[\lg(n!)^3]+1$, număr care este egal cu `3lgn!+1`. Există mai multe posibilități pentru calculul valorii `lgn!`, cea mai rapidă fiind transformarea logaritmului zecimal în logaritm natural și folosirea *formulei lui Stirling*. Formula de transformare din logaritm zecimal în logaritm natural este $\lg N=\frac{\ln N}{\ln 10}$, iar formula lui Stirling este:

$$\ln(n!)=\left(n+\frac{1}{2}\right)\cdot\ln n-n+\sigma+\frac{1}{12n}-\frac{1}{360n^3}+O\left(\frac{1}{n^5}\right),$$

unde $e^{\sigma}=\sqrt{2\pi}$, adică $\sigma=\ln\sqrt{2\pi}=\frac{1}{2}\ln 2\pi$, iar $O\left(\frac{1}{n^5}\right)$ este un termen care evident tinde la 0, deci poate fi considerat nul (în cazul nostru erorile vor fi nesemnificative). În concluzie numărul de cifre al permanentului matricei *A* va fi:

$$Nr\_cifre=3\cdot\left[\frac{\left(n+\frac{1}{2}\right)\cdot\ln n-n+\frac{1}{2}\ln(2\pi n)+\frac{1}{12n}-\frac{1}{360n^3}}{\ln 10}\right]+1.$$

Listing PERMANEN.PAS

```
Program permanent;
var n,log_nat,log_zec:Real;
    f:Text;
Begin
  Assign(Input,'PERM.IN'); Reset(Input);
  Readln(n);
  Close(Input);
  Assign(Output,'PERM.OUT'); Rewrite(Output);
  log_nat:=(n+0.5)*Ln(n)-n+0.5*Ln(2*Pi)+
           1/(12*n)-1/(360*n*n*n){+...};
  log_zec:=log_nat/Ln(10);
  Writeln(Int(3*log_zec)+1:0:0);
  Close(Output)
End.
```

P079909: Pitagora

Pentru ca trei numere `a`, `b` și `c` (`a < b < c`) să poată fi laturile unui triunghi dreptunghic ele trebuie să satisfacă relația $a^2+b^2=c^2$ (teorema lui *Pitagora*). Cunoscând numărul `a` trebuie să determinăm perechi de numere `b` și `c` pentru care $a^2=c^2-b^2$, adică $a^2=(c+b)(c-b)$. Datorită faptului că `a`, `b` și `c` sunt numere naturale, înseamnă că `(c-b)` va trebui să fie divizor al lui a^2. Pentru a determina perechile cerute vom considera toți divizorii `d` ai lui a^2 și vom rezolva sistemul de două ecuații liniare:

$$\begin{cases}c-b=d\\ c+b=\frac{a^2}{d}\end{cases}$$

Rezolvând sistemul obținem $c=\frac{a^2+d^2}{2d}$ și $b=\frac{a^2-d^2}{2d}$.

La sfârșit va trebui să verificăm dacă numerele `b` și `c` obținute în acest mod satisfac condiția suplimentară `a<b<c`.

Listing: PITAGORA.PAS

```
Program Pitagora;
var a,b,c:Longint;
    f:Text;

procedure Citire;
begin
  Assign(f,'PITAGORA.IN'); Reset(f);
  Readln(f,a);
  Close(f)
end;

procedure Rezolvare;
var d:Longint;
begin
  Assign(f,'PITAGORA.OUT'); Rewrite(f);
  if Odd(a) then d:=1 else d:=2;
  while d<=a do
  begin
    if (a*a mod d=0)
    then
    begin
      c:=(a*a div d+d) div 2;
      b:=c-d;
      if (a<b) and (a*a=(c-b)*(b+c))
      then Writeln(f,b,' ',c)
    end;
    Inc(d,2)
  end;
  Close(f)
end;

Begin
  Citire;
  Rezolvare
End.
```

CONCURS

Prima etapă a concursului organizat de Gazeta de Informatică s-a încheiat; în acest număr vă vom prezenta punctajele primilor clasaţi în acest concurs, soluţiile problemelor de la prima rundă, precum şi cele trei probleme ale rundei a treia.

Punctaj maxim

Spre surprinderea noastră au existat doar şase concurenţi care au obţinut punctaj maxim la prima etapă a concursului. Aceştia sunt:

Adrian Cârcu, Bistriţa
Gabriel Erzse, Oradea
Cosmin Silvestru Negruseri, Bistriţa
Mihai Pătraşcu, Craiova
Adrian Sándor, Bucureşti
Virgil Şerbănuţă, Bucureşti

Descalificări

Din nefericire unii concurenţi nu au luat în considerare prevederea regulamentară potrivit căreia redacţia poate hotărî descalificarea anumitor participanţi în cazul în care aceştia trimit soluţii asemănătoare. Datorită faptului că au existat surse care se diferenţiau numai prin denumirea variabilelor, comitetul de organizare a concursului a hotărât descalificarea următorilor concurenţi:

Mihai Hărănguş, Cluj
Danciu Maxim, Cluj
Andrei Dănuţ Cioban, Cluj

Aşteptarea reacţiei

O mulţime de programe aşteptau apăsarea unei taste înaintea terminării execuţiei. Acest fapt nu putea duce decât la depăşirea limitei de timp admise pentru execuţia programului şi la depunctarea concurenţilor chiar dacă, de altfel, programele ar fi funcţionat corect în mod normal.

Erori de compilare

Au existat o mulţime de cazuri în care sursele Pascal trimise nu puteau fi compilate, eroarea apărută fiind:

```
Error 116: Must be in 8087 mode to compile
           this.
```

Totul s-ar fi rezolvat prin includerea directivei de compilare `{$N+}` la începutul programului, dar probabil că mediul Pascal era în aşa fel setat pe calculatoarele pe care au lucrat respectivii concurenţi încât modul 8087 era implicit activat. Totuşi în mod normal modul 8087 nu este activat şi cum, potrivit regulamentului, nu sunt admise opţiuni de compilare speciale, directiva respectivă trebuia să apară în cadrul sursei dacă se dorea folosirea unor tipuri de date care necesită modul 8087.

Erori de execuţie

Nu au fost rare cazurile în care nu au fost respectate denumirile fişierelor indicate în enunţ sau s-a considerat că aceste fişiere se află într-un anumit director, altul decât cel curent. Datorită faptului că în enunţ nu era indicat directorul în care se află aceste fişiere, este evident că aceste fişiere se află în directorul curent.

Termen

Rezolvările problemelor publicate în acest număr pot fi trimise până la data de:

20 Ianuarie 2000.

Clasament GInfo Top 50

1	Adrian Cârcu, Bistriţa	300
1	Adrian Sándor, Bucureşti	300
1	Gabriel Erzse, Oradea	300
1	Mihai Pătraşcu, Craiova	300
1	Cosmin Negruseri, Bistriţa	300
1	Virgil Şerbănuţă, Bucureşti	300
7	Silviu Bota, Baia Mare	288
8	Ioana Ileană, Alba Iulia	285
9	Marius George Andrei, Bucureşti	280
9	Daniel Dumitran, Bucureşti	280
11	Valentin Muste, Baia Mare	268
12	Cristian Cadar, Bucureşti	260
13	Cristian Alexandrescu, Botoşani	245
14	Horea-Adrian Haitonic, Năsăud	235
14	Vlad Vâlceanu, Arad	235
16	Lucian Burja, Blaj	228
17	Gabriel Pârvan, Râmnicu Sărat	225
18	Arcadie Crăcan, Bălţi	220
18	Ionuţ Trifan, Buzău	220
20	Anghel Marian, Buftea	218
21	Ileana Constantinescu, Sibiu	213
21	Marius Neagu, Bucureşti	213
23	Florin Sabău, Reşiţa	208
24	Adrian Bălan, Târgu Jiu	205
25	Andrei Meliş, Bucureşti	204
26	Csaba Patcas, Oradea	200
26	Daniel Popa, Orăştie	200
26	Radu Streşina, Sinaia	200
26	Pál Szász, Marghita	200
30	Radu Călin Stanca, Geoagiu	188
30	Ciprian Baicu, Dr. Tr. Severin	188
32	Horea Coroiu, Cluj-Napoca	186
32	Marius Muja, Orăştie	186
32	Paul Rusu, Cluj-Napoca	186
35	Andrei Csibi, Cluj-Napoca	185
36	Tudor-Andrei Tataranu, Buzău	183
37	Cosmin Mărginean, Blaj	181
37	Valentin Potor, Târgu Mureş	181
37	Daniel Bolohan, ***	181
40	Robert Baban, Baia Mare	180
40	Andrei David, Râmnicu Vâlcea	180
42	Liviu Păunescu, Constanţa	178
43	Valentin Dinu, Piteşti	176
44	Victor Asavei, Dr. Tn. Severin	172
44	Alexandru Măscăsan, Tg. Mureş	172
46	Robert Badita, Craiova	171
46	Sergiu Bursuc, Telenești	171
46	Mihai Maerean, Slobozia	171
46	Daniel Porumbel, Buzău	171
46	Mădălin Răduţă, Parscov	171

Runda 1, problema 1

Volumul piramidei

Aceasta a fost cea mai ușoară problemă din concurs, dovadă și faptul că au existat mulți concurenți care au obținut punctaj maxim. Punctajul mediu obținut de concurenți pentru această problemă a fost de 52.29 puncte, un punctaj destul de scăzut având în vedere gradul de dificultate al problemei.

Pentru rezolvarea acestei probleme erau necesare cunoștințe minime de geometrie analitică în spațiu și anume formula volumului tetraedrului determinat de patru puncte din spațiu, ale căror coordonate se cunosc:

$$V = \frac{1}{6} \cdot [\Delta_3] \text{ unde } \Delta_3 = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{bmatrix}.$$

Există și o formulă generalizată pentru spații cu n dimensiuni, atunci când se cunosc coordonatele a n+1 puncte. Formula hipervolumului de ordinul n al unui corp n-dimensional este:

$$V = \frac{1}{n!} \cdot [\Delta_n] \text{ unde } \Delta_n = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^n & 1 \\ x_2^1 & x_2^2 & \cdots & x_2^n & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^1 & x_n^2 & \cdots & x_n^n & 1 \\ x_{n+1}^1 & x_{n+1}^2 & \cdots & x_{n+1}^n & 1 \end{bmatrix}.$$

Problema putea fi rezolvată și dacă nu se cunoștea această formulă folosind formula clasică potrivit căreia volumul unui tetraedru este o treime din produsul dintre aria bazei și înălțimea tetraedrului, dar pentru aceasta trebuiau efectuate multe operații cu numere reale care ar fi scăzut precizia rezultatului. O soluție interesantă furnizată de unii concurenți a fost rotirea piramidei până când una dintre fețe devine paralelă cu planul orizontal și apoi determinarea simplă a ariei bazei și a înălțimii. Este evident că o astfel de rezolvare duce la rezultatul corect deoarece prin rotirea unui corp, volumul său nu se modifică.

A surprins numărul ridicat al celor care nu au fost atenți la enunțul problemei unde se preciza clar că cele patru puncte din spațiu au coordonate reale, foarte mulți concurenți rezolvând corect problema din punct de vedere logic, dar considerând că cele patru puncte au coordonate întregi. Din nefericire nici unul dintre cele douăzeci de teste cu ajutorul cărora au fost testate programele nu conțineau puncte având coordonate întregi, motiv pentru care respectivii concurenți nu au obținut nici un punct la această problemă.

Au existat concurenți care au uitat că volumul se calculează folosind valoarea absolută a determinantului Δ_3 și au obținut volume negative în unele cazuri. Din aceste motive acești concurenți au obținut aproximativ 50% din punctajul total, datorită unei greșeli minore.

Vă prezentăm în continuare soluția oficială pentru această problemă:

Listing: VOLUM.CPP

```
#include <stdio.h>
#include <math.h>
#define EPSILON 1E-9

double det3(double a11,double a12,double a13,
            double a21,double a22,double a23,
            double a31,double a32,double a33){
  return a11*a22*a33+a21*a32*a13+a31*a12*
    a23-a11*a23*a32-a12*a21*a33-a13*a22*a31;
}

void main(void){
  FILE *f=fopen("VOLUM.IN","rt");
  double a[4][3],vol;
  int i,j;
  for (i=0;i<4;i++)
    for (j=0;j<3;j++)
      fscanf(f,"%lf",&a[i][j]);
  fclose(f);
  f=fopen("VOLUM.OUT","wt");
  vol=fabsl(det3(a[0][0],a[0][1],a[0][2],
                 a[1][0],a[1][1],a[1][2],
                 a[2][0],a[2][1],a[2][2])-
            det3(a[0][0],a[0][1],a[0][2],
                 a[1][0],a[1][1],a[1][2],
                 a[3][0],a[3][1],a[3][2])+
            det3(a[0][0],a[0][1],a[0][2],
                 a[2][0],a[2][1],a[2][2],
                 a[3][0],a[3][1],a[3][2])-
            det3(a[1][0],a[1][1],a[1][2],
                 a[2][0],a[2][1],a[2][2],
                 a[3][0],a[3][1],a[3][2]))/6;
  if (vol<EPSILON) fprintf(f,"0");
  else fprintf(f,"%.3lf",vol);
  fclose(f);
}
```

Au rezolvat corect:

Cristian Alexandrescu, Botoşani
Cătălin Alexandru, Bucureşti
Marius George Andrei, Bucureşti
Liviu Andreicuţ, Bucureşti
Marian Anghel, Buftea
Victor Asavei, Drobeta Turnu-Severin
Robert Baban, Baia Mare
Ciprian Baicu, Dr. Tn. Severin
Robert Bădiţă, Craiova
Adrian Bălan, Târgu Jiu
Daniel Bolohan, ***
Silviu Bota, Baia Mare
Marius Botiş, Baia Mare
Lucian Burja, Blaj
Sergiu Bursuc, Teleneşti
Radu Butcovan, Orăştie
Cristian Cadar, Bucureşti
Adrian Cârcu, Bistriţa
Andrei Clocotici, Iaşi
Ileana Constantinescu, Sibiu
Horea Coroiu, Cluj-Napoca
Arcadie Crăcan, Bălţi
Andrei Csibi, Cluj-Napoca
Andrei David, Râmnicu Vâlcea
Dorin Cristian Dinca, Slatina
Mihai Dondera, Bucureşti
Ciprian Duduială, Cluj-Napoca
Daniel Dumitran, Bucureşti
Gabriel Erzse, Oradea
Aurel Mihai Fulger, Constanţa
Horea-Adrian Haitonic, Năsăud
Ioana Ileană, Alba Iulia
Codruţ-Lucian Lazăr, Câmpeni
Mihai Maerean, Slobozia
Cosmin Mărginean, Blaj
Alexandru Măscăsan, Târgu Mureş
Andrei Meliş, Bucureşti
Alexandru Motriuc, Braşov
Marius Muja, Orăştie
Valentin Muste, Baia Mare
Marius Neagu, Bucureşti
Ion Nechita, Slatina
Cosmin Silvestru Negruseri, Bistriţa
George Stelian Pârcăbălescu, Slatina
Gabriel Pârvan, Râmnicu Sărat
Mihai Pătraşcu, Craiova
Liviu Păunescu, Constanţa
Daniel Popa, Orăştie
Daniel Porumbel, Buzău
Valentin Potor, Târgu Mureş
Mădălin Răduţă, Parscov
Andrei Mihnea Răileanu, Slatina
Paul Rusu, Cluj-Napoca
Cosmin Adrian Rusu, Buzău
Florin Sabău, Reşiţa
Adrian Sándor, Bucureşti
Mihai Sămărghitan, Târgu Mureş
Radu Călin Stanca, Geoagiu
Radu Streşina, Sinaia
Pál Szász, Marghita
Virgil Şerbănuţă, Bucureşti
Ionuţ Trifan, Buzău
Vlad Vâlceanu, Arad
Călin Voichiţa, Cluj-Napoca

Runda 1, problema 2

Dansul fluturilor (partiții cu bucluc)

Se pare că aceasta a fost cea mai dificilă problemă a etapei, existând foarte puțini concurenți care au obținut punctaj maxim. Punctajul mediu obținut de concurenți pentru această problemă a fost de 20.10 puncte, un punctaj destul de scăzut deoarece problema nu era foarte dificilă.

Această problemă a reprezentat un fel de test de fidelitate pentru cititorii *GInfo*. În numărul 8/8 al GInfo a fost prezentată o metodă "originală" folosită de membrii lotului național de informatică din 1998 pentru a rezolva problema "*Rigla Golomb*" propusă spre rezolvare în cadrul taberei de pregătire de la Constanța. Această metodă poate fi aplicată în cazul în care cardinalul mulțimilor soluțiilor este suficient de mic pentru ca soluțiile să poată fi memorate.

În cazul acestor probleme existau aproximativ `1000` de variante pentru datele de intrare, deci existau cel mult `1000` de soluții posibile. Rezultatul corect în cazul în care există `1000` de fluturi este:

`151638555895672880865922417102336560`0,

un număr de `37` de cifre. Deci, pentru a memora toate soluțiile posibile, avem nevoie de aproximativ `37 KB` de memorie.

În concluzie o metodă de rezolvare ar fi fost determinarea celor `1000` de soluții posibile prin diferite metode și construirea unui tablou de `1000` de șiruri de caractere care să conțină aceste soluții.

Totuși, determinarea celor `1000` de soluții posibile nu era o sarcină foarte ușoară. După o scurtă analiză ne dăm seama că problema se reduce la determinarea unei partiții a numărului `n` care are proprietatea că numerele care formează partiția au cel mai mare cmmmc.

Este evident că o astfel de partiție nu poate conține decât numere prime, puteri de numere prime și, eventual, numărul 1.

Au existat și concurenți care au rezolvat problema folosind metode "tradiționale" cum ar fi *backtracking* și programare dinamică, unii dintre ei reușind să obțină punctaje destul de mari folosind aceste metode.

Una dintre cele mai interesante erori care apărea era omiterea posibilității apariției numărului 1 în cadrul partiției; astfel se obținea pentru un anumit număr de fluturi un rezultat care era mai mic decât rezultatele obținute pentru mai puțini fluturi. Este evident că dacă pentru `n` fluturi dansul durează `T` secunde, atunci pentru `n+1` fluturi dansul va dura cel puțin `T` secunde. Într-adevăr, dacă vom considera că noul fluture va rămâne tot timpul în căsuța lui atunci dansul celorlalți fluturi va dura `T` secunde, deci prin adăugarea unui fluture, dansul nu va dura mai puțin. De exemplu, pentru `19` fluturi dansul va dura `420` de secunde (`19=3+4+5+7`, `[3,4,5,7]=420`), deci pentru `20` de fluturi dansul va dura cel puțin `420` de secunde. Dacă nu am lua în considerare și numărul `1`, pentru `20` de fluturi am fi obținut `210` secunde (`20=10+7+3`, `20=7+6+5+2` etc.).Totuși partiționarea cea mai bună este `20=1+3+4+5+7`; de asemenea, pentru `21` de fluturi vom avea `420` de secunde corespunzătoarea partiției `21=1+1+3+4+5+7`.

Au rezolvat corect:

Silviu Bota, Baia Mare

Adrian Cârcu, Bistrița

Gabriel Erzse, Oradea

Cosmin Silvestru Negruseri, Bistrița

Csaba Patcas, Oradea

Mihai Pătrașcu, Craiova

Adrian Sándor, București

Virgil Șerbănuță, București

Vă prezentăm în continuare soluția oficială pentru această problemă:

Listing FLUTURI.CPP

```
#include <stdio.h>

void main(void){
  int n;
  FILE *f=fopen("FLUTURI.IN","rt");
  fscanf(f,"%d",&n);
  fclose(f);
  f=fopen("FLUTURI.OUT","wt");
  switch (n){
  case 2:
    fprintf(f,"2");
    break;
  case  3:
    fprintf(f,"3");
    break;
  case  4:
    fprintf(f,"4");
    break;
  case  5:
    fprintf(f,"6");
    break;
  case  6:
    fprintf(f,"6");
    break;
  case  7:
    fprintf(f,"12");
    break;
  case  8:
    fprintf(f,"15");
    break;
  case  9:
    fprintf(f,"20");
    break;
  case 10:
    fprintf(f,"30");
    break;
  //...
  case 996:
    fprintf(f,"144330673683833223956721336760055280O");
    break;
  case 997:
    fprintf(f,"144330673683833223956721336760055280O");
    break;
  case 998:
    fprintf(f,"144330673683833223956721336760055280O");
    break;
  case 999:
    fprintf(f,"144330673683833223956721336760055280O");
    break;
  case 1000:
    fprintf(f,"151638555895672880865922471023365600");
  }
  fclose(f);
}
```

Info...

Numere prime

Probabil că în rezolvarea acestei probleme ați avut nevoie de un algoritm de generare a numerelor prime. Vă prezentăm în continuare o modalitate mai simplă de determinare a numerelor prime care se bazează pe următoarea teoremă: *Toate numerele impare mai mari decât 1 sunt prime. Pentru* a demonstra această teoremă am apelat la un matematician, un biolog, un fizician, un chimist și un informatician.

Matematicianul a studiat problema și a spus: *3 este număr prim, 5 este număr prim, 7 este număr prim; facem repede o inducție și iese imediat.*

Biologul nu s-a gândit prea mult și a zis: *3 este număr prim, 5 este număr prim, 7 este număr prim, 9 este număr prim, 11 este număr prim, 13 este număr prim, 15 este număr prim, 17 este număr prim, 19 este număr prim, 21 este număr prim și nu mai are rost să continuăm deoarece toate numerele impare sunt prime.*

Fizicianul a început și el: *3 este număr prim, 5 este număr prim, 7 este număr prim, 9 - eroare experimentală, 11 este număr prim, 13 este număr prim, deci toate numerele impare mai mari decât 1 sunt prime.*

Chimistul, după un moment de gândire, a spus: *3 este număr prim, 5 este număr prim, 7 este număr prim, iar chimiștii nu lucrează cu numere mai mari deci, din punctul nostru de vedere, toate numerele impare sunt prime.*

Informaticianul, mai isteț, a spus imediat: *cel mai simplu e să facem un program care să verifice dacă toate numerele impare mai mari decât 1 sunt prime.* După 10 minute, a lansat programul în execuție și rezultatul nu a întârziat să apară pe ecran: *3 este număr prim, 5 este număr prim, 7 este număr prim, 7 este număr prim, 7 este număr prim, 7 este număr prim...*

Runda 1, problema 3

Hambar

Această problemă, chiar dacă era ușor de rezolvat, i-a pus în dificultate pe mulți dintre concurenți. Totuși punctajul mediu obținut de concurenți a fost de 52.15 puncte, aproximativ egal cu cel obținut pentru problema determinării volumului tetraedrului, considerată mult mai ușoară.

În enunțul acestei probleme a apărut o mică eroare în sensul că deși se preciza că fișierele de intrare și ieșire sunt `HAMBAR.IN` și `HAMBAR.OUT`, în cadrul exemplului apăreau fișierele `INPUT.TXT` și `OUTPUT.TXT`; evident, au fost considerate corecte ambele variante.

Problema se rezolvă foarte simplu folosind metoda programării dinamice. Vom păstra o matrice pătratică de dimensiune `N`, fiecare element al matricei indicând latura maximă a hambarului care are colțul din dreapta-jos în parcela corespunzătoare elementului respectiv.

Această matrice poate fi completată astfel: se pleacă din colțul din stânga-sus și pentru fiecare punct `(i,j)` care nu corespunde unei parcele cu pomi valoarea A_{ij} este obținută prin formula $\min(A_{i-1,j}, A_{i,j-1}, A_{i-1,j-1})+1$; în cazul în care avem un pom în parcela `(i,j)` A_{ij} va primi valoarea `0`.

Formula anterioară este determinată de faptul că hambarul care conține noua parcelă va conține, în principiu, și cele trei parcele din punctele de coordonate `(i-1,j)`, `(i,j-1)`, `(i-1,j-1)`. Evident, dimensiunea hambarului corespunzător acestei parcele va depinde de dimensiunea hambarului corespunzător celor trei parcele și va trebui să alegem valoarea minimă pentru că altfel hambarul corespunzător parcelei `(i,j)` ar ocupa și o parcelă cu pomi. Evident, adăugăm 1 la acest minim pentru că dimensiunea hambarului crește cu 1 dacă conține și parcela `(i,j)`.

Rezultatul va fi dat de elementul maxim al matricei `A`. Datorită faptului că acest maxim a fost obținut printr-o singură parcurgere a matricei, ordinul de complexitate al algoritmului prezentat este $O(n^2)$.

Există multe alte metode de rezolvare, bazate pe diferite idei având complexități variabile. Datorită limitelor impuse pentru datele de intrare un algoritm având complexitatea $O(n^3)$ s-ar fi încadrat în timp.

Una dintre cele mai frecvente greșeli a fost netratarea cazului în care fiecare parcelă conține pomi, deci nu poate fi construit nici un hambar. În acest caz lungimea maximă a laturii hambarului este `0` și nu `1` cum au indicat unii concurenți.

Vă prezentăm în continuare soluția oficială pentru această problemă:

Au rezolvat corect:

Cristian Alexandrescu, Botoșani
Marius George Andrei, București
Marian Anghel, Buftea
Adrian Bălan, Târgu Jiu
Rodica Bălașa, Rmânicu Vâlcea
Lucian Burja, Blaj
Cristian Cadar, București
Ciprian Cană, Vaslui
Adrian Cârcu, Bistrița
Ileana Constantinescu, Sibiu
Arcadie Crăcan, Bălți
Daniel Dumitran, București
Gabriel Erzse, Oradea
Horea-Adrian Haitonic, Năsăud
Ioana Ileană, Alba Iulia
Marius Neagu, București
Cosmin Silvestru Negruseri, Bistrița
Csaba Patcas, Oradea
Mihai Pătrașcu, Craiova
Daniel Popa, Orăștie
Florin Sabău, Reșița
Adrian Sándor, București
Radu Streșina, Sinaia
Virgil Șerbănuță, București
Tudor-Andrei Tătăranu, Buzău
Ionuț Trifan, Buzău
Vlad Vâlceanu, Arad

Listing HAMBAR.CPP

```
#include <stdio.h>

char a[200][200],n,t,max;

void lecture(void){
   int i,x,y;
   FILE *f=fopen("HAMBAR.IN","rt");
   fscanf(f,"%d%d",&n,&t);
   for (i=0;i<t;i++)
     fscanf(f,"%d%d",&x,&y),a[x-1][y-1]=-1;
   fclose(f);
}

int min(int a,int b){
   return (a<b)?a:b;
}

void solution(void){
   int i,j;
   max=0;
   for (i=0;i<n;i++)
      for (j=0;j<n;j++)
         if (a[i][j]!=-1){
            a[i][j]=min(a[i-1][j],
                        min(a[i][j-1],a[i-1][j-1]))+1;
            if (a[i][j]>max)
               max=a[i][j];
         }
         else a[i][j]=0;
}

void ecriture(void){
   FILE *f=fopen("HAMBAR.OUT","wt");
   fprintf(f,"%d",max);
   fclose(f);
}

void main(void){
   lecture();
   solution();
   ecriture();
}
```

Erată

În exemplele prezentate pentru problema 2 (Spiriduși) de la etapa a doua a concursului Bursele Agora s-au "strecurat" două greșeli. De fapt exemplele sunt:

```
SPRITES.IN
4 4
SPRITES.OUT
35
```

respectiv

```
SPRITES.IN
5 5
SPRITES.OUT
126
```

Ne cerem scuze pentru situația creată și vă promitem că vom încerca să facem în așa fel încât o astfel de greșeală să nu mai apară.

Termenul limită pentru trimiterea rezolvărilor problemelor propuse la Runda 2 se amân până pe data de

31 Decembrie 1999

Succes în continuare!

Date personale

Au existat mai mulți concurenți care nu au precizat toate datele personale necesare pentru a putea fi înscriși în concurs. Toți aceștia au fost înscriși provizoriu, redacția așteptând trimiterea acestor date până la data de 31 Ianuarie 2000. Concurenții care nu vor furniza toate datele cerute în timp util vor fi descalificați.

Runda 3

CR3P1 Blocaje

În *AgoraLand* există n orașe legate între ele prin m străzi cu două sensuri de circulație. Să se determine un număr minim de străzi care trebuie blocate pentru ca să existe cel puțin o pereche de orașe (A,B) în *AgoraLand* astfel încât plecând din orașul A să nu existe nici o posibilitate de a ajunge în orașul B. Orașele din *AgoraLand* sunt identificate prin numere întregi cuprinse între 1 și n și nu pot exista mai multe străzi între două orașe.

Date de intrare

Fișierul **BLOCAJ.IN** conține pe prima linie numerele m și n separate printr-un singur spațiu și pe următoarele m linii câte o pereche de numere întregi x și y cu semnificația: *"Există o stradă care leagă orașele x și y"*.

Date de ieșire

Fișierul **BLOCAJ.OUT** trebuie să conțină pe prima linie numărul nr al străzilor care trebuie blocate și pe următoarele nr linii câte o pereche de numere x și y cu semnificația: *"Strada care leagă orașele x și y trebuie blocată"*.

Exemplu

BLOCAJ.IN	BLOCAJ.OUT
8 14	2
1 2	2 5
1 3	3 8
1 4	
2 3	
2 4	
2 5	
3 4	
3 8	
5 6	
5 7	
5 8	
6 7	
6 8	
7 8	

Timp de execuție: 1 secundă/test

CR3P2 Numere prime

Dându-se o mulțime de numere întregi care au cel mult 1000 de cifre să se determine care dintre acestea sunt prime și care nu.

Numere (cel mult 10) se află în fișierul **PRIME.IN** (câte unul pe o linie), iar răspunsurile (DA sau NU) se vor afla în fișierul **PRIME.OUT** (câte un răspuns pe o linie). În cazul în care unul dintre răspunsuri este incorect nu se va nici un punct pentru testul respectiv.

Exemplu

PRIME.IN	PRIME.OUT
1	NU
2	DA
3	DA
4	NU
5	DA
6	NU
7	DA
8	NU
9	NU
10	NU

Timp de execuție: 10 secunde/test

CR3P3 Joc

Se dă o tablă dreptunghiulară de dimensiuni m×n. Jocul constă în așezarea în căsuțele tablei a unor numere între 0 și 8 astfel încât valoarea elementelor unei căsuțe să fie egală cu numărul vecinilor care au valori impare. Numărul 0 este considerat JOKER, adică poate fi așezat într-o căsuță chiar dacă numărul vecinilor impari nu este 0.

Punctajul unei table este dat de suma valorilor din căsuțele sale, din care se scade 10% pentru fiecare JOKER folosit. (Dacă veți folosi 2 JOKERS scăderea nu va fi cu 20%, ci cu 19%, adică se va scădea "10% din ce a rămas"). De exemplu, dacă o tablă are suma elementelor 1000 și folosește 3 JOKERS atunci se va scădea 10%=100 puncte, din cele 900 de puncte rămase se vor scădea 10%=90 puncte, iar din cele 810 puncte rămase se vor mai scădea 10%=81 puncte, punctajul tablei devenind 729 puncte. Să se determine o tablă având cel mai mare punctaj posibil.

Date de intrare

Fișierul **JOC.IN** va conține numerele m și n separate printr-un singur spațiu (1≤m, n≤500).

Date de ieșire

Fișierul **JOC.OUT** va conține m linii pe fiecare aflându-se câte n numere separate printr-un singur spațiu; aceste numere vor reprezenta valorile atașate căsuțelor de pe tablă

Exemplu

JOC.IN	JOC.OUT
3 3	3 5 3
	3 5 3
	2 0 2

Punctajul tablei este 23.4 puncte

Timp de execuție: 1 secundă/test
Se acordă punctaje parțiale

Introducere în teoria GRAFURILOR

Cristian Cadar

O teorie pe care dacă o știi, ți se pare foarte frumoasă, rezolvarea de probleme, folosind modele din teoria grafurilor ți se pare ușoară. În caz contrar, poate fi un coșmar. Și cum noi vrem să "sărim" în ajutorul vostru...

Noțiuni introductive

Definiție

Se numește ***graf neorientat*** *o pereche ordonată* G=(X,U), *unde* X *este o mulțime finită și nevidă de elemente numite noduri sau vârfuri, iar* U *este o mulțime de perechi de elemente ale lui* X, *numite muchii. În cazul în care perechile din* U *sunt ordonate, acestea se numesc arce și graful se va numi orientat. Vom nota cu* card X=n *și* card U=m. *De asemenea, nodurile grafului vor fi numerotate cu numere consecutive de la* 1 *la* n.

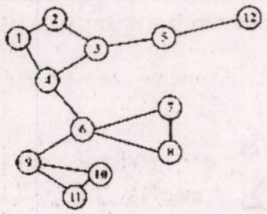


Fig. 1: Reprezentarea unui graf neorientat

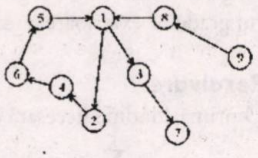


Fig. 2: Reprezentarea unui graf orientat

În figurile 1 și 2 avem reprezentarea grafică a unui graf neorientat și respectiv a unuia orientat. Această reprezentare grafică asociază fiecărui nod al grafului un cerc în care este precizat numărul vârfului, iar fiecărei muchii o linie dreaptă sau curbă care unește vârfurile corespunzătoare. În cazul grafurilor orientate, arcele vor fi reprezentate ca și muchiile, doar că în plus vor avea asociate săgeți care vor indica sensul lor.

Grafurile au numeroase aplicații practice, ele fiind folosite pentru a reprezenta relațiile dintre diferite tipuri de elemente. Un exemplu clasic de folosire a grafurilor este cel al memorării orașelor dintr-o anumită regiune și a drumurilor care le leagă. Bineînțeles, orașele vor reprezenta vârfurile grafului, iar drumurile dintre ele vor reprezenta muchiile sau arcele grafului (în funcție de restricțiile de sens unic existente).

Definiție

Un ***graf parțial*** *al grafului* G=(X,U) *este un graf* G'=(X,U') *astfel încât* U'⊆U.

Deci, un graf parțial al unui anumit graf se obține prin eliminarea a zero sau mai multe muchii ale grafului. În figura 3 este reprezentat un graf parțial al grafului din figura 1.

Definiție

Un ***subgraf*** *al grafului* G=(X,U) *este un graf* H=(Y,V) *astfel încât* Y⊆X *și* V *conține toate muchiile din* U *care unesc vârfuri din* Y. *În figura 4 este reprezentat un subgraf al grafului din figura 2.*

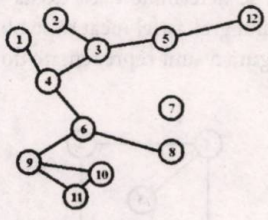


Fig. 3: Graf parțial al grafului din figura1

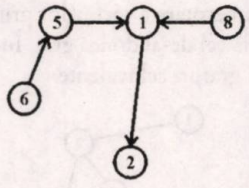


Fig. 4: Subgraf al grafului din figura 2

În cele ce urmează vom determina numărul grafurilor neorientate cu n vârfuri. Numărul maxim de muchii pe care îl poate avea un graf neorientat este de C_n^2. Pentru a forma toate grafurile neorientate vom considera pe rând toate submulțimile de muchii din mulțimea totală de C_n^2 muchii. Numărul de astfel de submulțimi este egal cu:

$$2^{C_n^2} = 2^{\frac{n(n-1)}{2}}$$

care este și numărul de grafuri neorientate cu n vârfuri. În mod similar se determină și numărul grafurilor orientate care este $2^{n(n-1)}$. Folosind același raționament, rezultă că numărul grafurilor parțiale ale unui graf este 2^m, (m reprezintă numărul muchiilor) iar numărul subgrafurilor unui graf este 2^n.

Definiție

Într-un graf neorientat, numărul muchiilor incidente cu un anumit vârf v reprezintă ***gradul vârfului*** *respectiv și se notează cu* d(v). *Într-un graf orientat numărul muchiilor care "ies" dintr-un vârf v reprezintă* ***gradul exterior*** *al vârfului respectiv și se notează cu* d_ext(v), *iar numărul muchiilor care "intră" într-un vârf v reprezintă* ***gradul interior*** *al vârfului respectiv și se notează cu* d_int(v).

Un vârf cu gradul 0 se numește *vârf izolat*, iar un vârf cu gradul 1 se numește *vârf terminal.*

Avem următoarele relații importante între gradele vârfurilor unui graf (x_1, x_2, ..., x_n sunt vârfurile grafului):

1) $0 \le d(x_i) \le n-1,\ i = \overline{1,n}$;

2) $\sum_{i=1}^{n} d(x_i) = 2m.$

deoarece fiecare muchie contribuie cu câte o unitate la gradele celor două vârfuri pe care le unește;

3) Numărul gradelor impare dintr-un graf este par - rezultă imediat din (2)

4) Un graf cu $n \ge 2$ vârfuri conține cel puțin două vârfuri cu același grad - rezultă din (2) și din principiul lui Dirichlet.

5) $\sum_{i=1}^{n} d\_int(x_i) = \sum_{i=1}^{n} d\_ext(x_i) = m.$

Problemă

Una dintre problemele care pot interveni constă în determinarea echivalenței a două grafuri: dându-se două grafuri (orientate sau neorientate) să se determine dacă există o numerotare a vârfurilor primului graf astfel încât să se obțină cel de-al doilea graf. În figura 5 sunt reprezentate două grafuri echivalente.

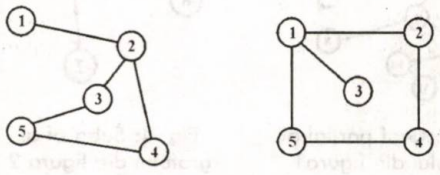


Fig. 5 Grafuri echivalente

Rezolvare

Rezolvarea acestei probleme este exponențială, se obține însă un timp foarte bun în felul următor: se determină șirul gradelor pentru cele două grafuri și se ordonează crescător aceste șiruri. Dacă după ordonare cele două șiruri nu sunt identice, atunci problema nu admite soluție. În caz contrar, se încearcă "potrivirea" celor două șiruri, prin interschimbarea succesivă a vârfurilor având același grad. Bineînțeles, cu cât numărul vârfurilor cu același grad este mai mic, cu atât algoritmul va fi mai rapid.

Definiție

Un șir $d=(d_1, d_2, ..., d_n)$ *se numește* ***șir grafic*** *dacă există un graf care să aibă ca grade numerele din acest șir.*

Problemă

Dându-se un șir $a=(a_1, a_2, ..., a_n)$, să se determine dacă acesta este un șir grafic.

Rezolvare

Mai întâi ordonăm descrescător șirul dat. Obținem șirul d: $d_1 \ge d_2 \ge ... \ge d_n$. Condițiile necesare și suficiente pentru ca șirul d să fie grafic sunt:

1) $d_1 \le n-1,\ d_n \ge 0$

2) $(d_2-1, d_3-1, ..., d_{d_1+1}-1, d_{d_1+2}, ..., d_n)$ să fie șir grafic.

Cea de-a doua condiție ne dă o metodă de construcție a grafului. Presupunem că $(d_2-1, d_3-1, ..., d_{d_1+1}-1, d_{d_1+2}, ..., d_n)$ este un șir grafic corespunzător unui graf cu vârfurile $x_2, x_3, ..., x_n$. Atunci prin adăugarea vârfului x_1 și a muchiilor (x_1, x_2), (x_1, x_3), ..., (x_1, x_{d_1+1}) obținem un graf cu șirul gradelor $(d_1, d_2, ..., d_n)$.

După fiecare eliminare a nodului cu grad maxim din graf, secvența obținută conform celei de-a doua condiții trebuie ordonată, deci complexitatea acestui algoritm va fi $n \cdot O(sortare) = O(n^2 \cdot \log n)$.

Problemă

Aceeași problemă pentru un graf orientat: se dau două șiruri $d\_int=(d\_int_1, d\_int_2, ..., d\_int_n)$ și $d\_ext=(d\_ext_1, d\_ext_2, ..., d\_ext_n)$. Să se determine dacă există un graf orientat cu șirul gradelor interioare d_int și cu șirul gradelor exterioare d_ext.

Rezolvare

O primă condiție necesară este ca

$$\sum_{i=1}^{n} d\_int_i = \sum_{i=1}^{n} d\_ext_i .$$

Dacă această condiție este îndeplinită, problema se rezolvă folosind un algoritm de calcul al cuplajului maxim într-un graf[1].

În scopul rezolvării problemei formăm rețeaua de transport din figura 6, în care nodurile grafului sunt dublate. În prima serie de noduri intră de la sursă (nodul s) arce având costul egal cu cel al gradului interior corespunzător nodului respectiv, iar din cea de-a doua serie de noduri ies spre destinație (nodul d) arce având costul egal cu cel al gradului exterior corespunzător vârfului respectiv. De la fiecare nod din prima serie de noduri ducem arce de cost 1 către toate nodurile din seria a doua de noduri. Acest fapt are semnificația că într-un nod v pot intra

[1] *Algoritmul de cuplaj maxim va fi prezentat într-un articol viitor al acestui serial*

maxim d_int(v) arce, pot ieși maxim d_ext(v) arce, iar între oricare două noduri poate exista cel mult un arc. Având această rețea construită, aplicăm un algoritm de cuplaj maxim. Dacă cuplajul maxim are valoarea

$$\sum_{i=1}^{n} d\_int_i$$

atunci problema are soluție, în caz contrar nu are.

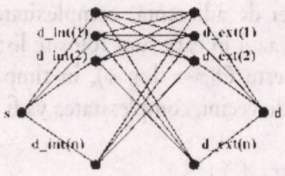


Fig. 6

Continuăm prezentarea mai multor clase speciale de grafuri:

Definiție

Se numește **graf complet** *cu* n *vârfuri un graf cu proprietatea că oricare două vârfuri sunt adiacente. Un graf complet cu* n *vârfuri se notează cu* K_n *și se referă în general la grafurile neorientate.*

În ceea ce privește grafurile orientate, un graf orientat este complet dacă între oricare două noduri x, y există fie un arc de la x la y, fie un arc de la y la x, fie ambele arce, deci pentru fiecare dintre cele C_n^2 perechi de noduri ale unui graf orientat cu n noduri, există trei posibilități ca aceste noduri să fie adiacente. Deci, există $3^{C_n^2} = 3^{\frac{n(n-1)}{2}}$ grafuri complete orientate cu n vârfuri.

Definiție

Se numește **graf turneu** *un graf orientat pentru care între oricare două vârfuri există un singur arc (într-un sens sau altul). Folosind același raționament ca și în cazul grafurilor complete, există* $2^{\frac{n(n-1)}{2}}$ *grafuri turneu cu* n *noduri.*

Grafurile turneu au următoarele proprietăți, care sunt foarte ușor de dedus:

1) $d\_int(x_i) + d\_ext(x_i) = n - 1, \forall i = \overline{1,n}$

2) $\sum_{i=1}^{n} d\_int(x_i) + \sum_{i=1}^{n} d\_ext(x_i) = C_n^2 = \frac{n(n-1)}{2}$

3) $\sum_{i=1}^{n} d\_int(x_i)^2 = \sum_{i=1}^{n} d\_ext(x_i)^2$

Definiție

Un graf G=(X,U) *se numește* **bipartit** *dacă mulțimea vârfurilor* X *poate fi partiționată în două mulțimi* Y *și* Z *(*$Y \cup Z = X$, $Y \cap Z = \emptyset$*) și orice muchie* $u \in U$ *are o extremitate în* Y *și una în* Z *(figura 7).*

Un graf bipartit este complet dacă pentru orice vârf y din Y și orice vârf z din Z, există muchia (x, z). Dacă card Y=p și card Z=q, atunci graful bipartit complet se notează cu $K_{p,q}$. Evident, numărul de muchii ale grafului bipartit $K_{p,q}$ este p×q.

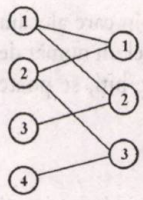


Fig. 7 Graf bipartit

Problemă

Dându-se un graf să se stabilească dacă este bipartit.

Rezolvare

O primă metodă, care presupune memorarea grafului, constă în parcurgerea grafului și atașarea alternantă a etichetelor 1 și 2 nodurilor parcurse (cu semnificația că nodul respectiv face parte din mulțimea Y respectiv mulțimea Z). Dacă se ajunge într-un nod vizitat cu o etichetă diferită de cea deja asociată, graful respectiv nu poate fi bipartit. Dacă nu se ajunge niciodată într-o astfel de situație, atunci graful este bipartit, construirea lui făcându-se pe baza etichetelor asociate.

O altă variantă, care nu necesită memorarea grafului - fiind astfel aplicabilă pentru grafuri de dimensiuni foarte mari - folosește modul de memorare al mulțimilor disjuncte, în care reuniunea a două mulțimi necesită timp constant, iar apartenența unui element la o mulțime necesită timp logaritmic. Folosind o astfel de structură, în momentul în care citim o muchie (x, y), găsim tatăl comun al celor două extremități. Dacă există două noduri pentru care lungimea drumului dintre ele (aflat ca suma lungimilor drumurilor de la tatăl comun la cele două vârfuri) este pară, atunci graful nu poate fi bipartit. În caz contrar el este bipartit, cele două mulțimi fiind cele rezultate în finalul algoritmului.

Definiție

Un graf pentru care între oricare două perechi de vârfuri pot exista oricât de multe muchii (sau arce) se numește **multigraf.**

Memorarea grafurilor

Matricea de adiacență

Cea mai cunoscută metodă de memorare a grafurilor neorientate este matricea de adiacență, definită în felul următor:

$$a(i,j) = \begin{cases} 1, \text{ dacă există muchie (arc) de la vârful } i \text{ la vârful } j \\ 0, \text{ în caz contrar} \end{cases}$$

Bineînțeles, în cazul grafurilor neorientate, această matrice este simetrică, folosindu-se efectiv numai jumătate din spațiul matricei. De aceea, în unele probleme este eficient ca cealaltă jumătate a matricei să fie folosită pentru reținerea altor informații.

Matricea de adiacență este folosită în general pentru grafuri cu un număr mic de noduri, deoarece dimensiunea ei este limitată de dimensiunea stivei. Deoarece stiva are aproximativ 65000 de bytes, vom putea aloca spațiu pentru o valoare aproximativă de cel mult 250 de noduri. To-

tuși, putem depăși această valoare în cazul în care alocăm matricea în heap, putând fi alocate în acest fel un număr de 800 de noduri. Mai mult chiar, lucrând pe biți, se poate ajunge la aproximativ 2200 de vârfuri.

Listele de muchii

Această metodă de memorare presupune reținerea unei matrice cu `2` linii și `m` coloane, pe fiecare coloană fiind reținute două extremități ale unei muchii.

Listele de muchii sunt folosite în cazul algoritmilor care prelucrează secvențial muchiile, cum ar fi de exemplu *algoritmul Kruskal* de aflare a arborelui parțial de cost minim. De asemenea, se recomandă folosirea acestei structuri în cazul *grafurilor rare* (cu număr mic de muchii).

Principala problemă a acestei structuri este faptul că găsirea unei muchii se face în complexitatea $O(m)$. Pentru a remedia acest fapt, se sortează muchiile în funcție de prima extremitate și în caz de egalitate, în funcție de cea de a doua, căutarea va deveni logaritmică (se folosește bineînțeles căutarea binară).

Listele de vecini

Listele de vecini sunt cele mai recomandate în tratarea algoritmilor din teoria grafurilor din două motive principale:

1) spațiul folosit este gestionat dinamic, putându-se memora astfel grafuri de dimensiuni mari;
2) complexitatea optimă pentru majoritatea algoritmilor fundamentali din teoria grafurilor (parcurgeri, conexitate, muchii și puncte critice, algoritmi de drum minim etc.) este obținută numai folosind liste de vecini.

Există două modalități de implementare a listelor de vecini. Prima dintre ele folosește o *matrice* `T` cu `2` linii și `2m` coloane și un vector `C` cu `n` elemente care reține pentru fiecare nod indicele coloanei din `T` pe care este memorat primul element al listei nodului respectiv (sau `0` dacă vârful respectiv nu are vecini). Apoi, pentru o anumită coloană `i` din `T`, `T(i,1)` reține un element din lista curentă, iar `T(i,2)` reține coloana pe care se găsește următorul element din lista respectivă sau `0` dacă lista s-a terminat.

Cea de-a doua implementare folosește pentru fiecare nod o *listă simplu înlănțuită* memorată în heap. De reținut că pentru fiecare listă este suficient să păstrăm o singură santinelă (cea de început a listei), introducerea vârfurilor făcându-se mereu la începutul listei (deoarece ordinea vârfurilor în listă nu contează).

În cazul în care numărul muchiilor este cunoscut, se recomandă primul mod de implementare. În cazul în care numărul muchiilor nu este cunoscut inițial, se recomandă cea de-a doua implementare (prima dintre ele fiind de multe ori în acest caz de neutilizat).

Parcurgerea grafurilor

Parcurgerea unui graf presupune examinarea în vederea prelucrării tuturor vârfurilor acelui graf într-o anumită ordine, ordine care să permită prelucrarea optimă a informațiilor atașate grafului.

Parcurgerea DF (parcurgerea în adâncime)

Parcurgerea DF (*Depth First*) presupune ca dintr-un anumit nod `v`, parcurgerea să fie continuată cu primul vecin al nodului `v` nevizitat încă. Cel mai cunoscut mod de implementare a parcurgerii DF se realizează cu ajutorul unei funcții recursive, dar există și cazuri în care este recomandată o implementare nerecursivă. De reținut că în cazul în care memorarea grafului se realizează prin intermediul matricei de adiacență, complexitatea parcurgerii este $O(n^2)$, în cazul în care sunt folosite listele de muchii complexitatea este $O(m+n \cdot \log m)$, în timp ce dacă sunt folosite listele de vecini, complexitatea va fi doar $O(m+n)$.

```
Parcurge_DF(nod k)
Vizitat(k) ← adevărat
i ← primul_vecin(k)
cât timp nu am parcurs toată lista vecinilor lui k
  dacă nu Vizitat(i)
    atunci Parcurge_DF(i)
  i ← următorul_vecin(k)
```

Parcurgerea BF (parcurgerea în lățime)

Parcurgerea BF (*Breath First*) presupune faptul că după vizitarea unui anumit nod `v`, sunt parcurși toți vecinii nevizitați ai acestuia, apoi toți vecinii nevizitați ai acestora din urmă, până la vizitarea tuturor nodurilor grafului.

Implementarea acestei metode se face folosind o coadă. Aceasta este inițializată cu un nod oarecare al grafului. La fiecare pas, se vizitează nodul aflat în vârful cozii și se adaugă în coadă toți vecinii nevizitați ai nodului respectiv.

Ca și în cazul parcurgerii DF, complexitatea optimă pentru această parcurgere, $O(m+n)$, se obține numai atunci când graful este memorat cu ajutorul listelor de vecini.

```
Parcurgere_BF(nod start)
Coada ← start
Vizitat(start) ← adevărat
cât timp coada nu este vidă
  ic ← nodul de la începutul cozii
  adaugă toți vecinii nevizitați ai lui ic în coadă
  Vizitat(i) ← adevărat, unde i este un vecin nevizitat al nodului ic
```

Parcurgerea BF vizitează vârfurile crescător în funcție de distanța de la nodul din care se pornește parcurgerea (*nodul start*). Astfel, dacă într-un graf cu costuri egale pe toate muchiile se dorește aflarea distanței minime dintre două vârfuri, se poate folosi parcurgerea BF. De asemenea, pentru un astfel de graf, se poate obține printr-o parcurgere BF un arbore pentru care suma costurilor de la vârfurile sale la nodul de start este minimă.

Cristian Cadar este student în anul II la Universitatea Politehnică București. Poate fi contactat prin e-mail la adresa ccadar@pcnet.pcnet.ro.

Pagini WEB cu JavaScript

Mihai Scorțaru

În acest episod vom prezenta o nouă clasă importantă pusă la dispoziție de limbajul JavaScript și anume clasa Date. De asemenea vom prezenta modul în care pot fi create diferite tipuri de ferestre folosind limbajul JavaScript.

Clasa Date

Obiectele instanță ale clasei *Date* sunt folosite pentru a obține informații referitoare la ceasul sistem de pe calculatorul vizitatorului paginii *Web*. Prin intermediul acestor obiecte pot fi determinate data și ora curentă, pot fi efectuate diferite operații cu date calendaristice sau momente ale zilei sau poate fi controlat modul în care este afișată pagina *Web* în funcție de informațiile furnizate de metodele acestor obiecte.

Există trei modalități de creare a obiectelor instanță ale clasei *Date*: inițializarea unui obiect cu data și ora curentă, inițializarea unui obiect prin indicarea datei sub forma numerică sau sub forma unui șir de caractere.

Pentru a crea un obiect care să conțină data și ora curentă secvența *JavaScript* corespunzătoare este:

```
Maintenant=new Date().
```

Dacă dorim să indicăm data sub formă numerică, secvența corespunzătoare este:

`unefois=new Date(`*zi, lună, ani, ore, minute, secunde*`).`

Un exemplu ar putea fi:

```
Anniversaire=new Date(6,8,79,18,10,0).
```

În cazul în care dorim să indicăm data sub forma unui șir de caractere putem folosi o secvență de genul:

`unefois=new Date("`*lună zi, an ore:minute:secunde*`").`

Un exemplu de utilizare este:

```
Aniversaire=new Date(August 6, 1979 18:10:00).
```

În *JavaScript* timpul este stocat în formatul de 24 de ore, orei 1p.m. corespunzându-i valoarea 13:00:00, orei 2p.m. valoarea 14:00:00 etc.

Indiferent de modul de creare a obiectului, pot fi omise informațiile referitoare la oră, implicit considerându-se că ora este 0:00:00.

Metodele obiectelor de tip dată

Metodele obiectelor de tip dată pot fi grupate în trei categorii: metode pentru preluare de informații, metode pentru setarea anumitor caracteristici și metode pentru conversie.

Metoda getDate()

Metoda `getDate()` poate fi folosită pentru a determina ziua din lună a datei stocate în obiectul pentru care este apelată; valoarea returnată este un număr cuprins între `1` și `31`. Următoarea secvență poate fi folosită pentru afișarea zilei curente:

```
Aujourdhui=new Date();
document.writeln(Aujourdhui.getDate());
```

Metoda getDay()

Metoda `getDay()` este folosită pentru a determina ziua din săptămână a datei stocate în obiectul pentru care este apelată; valoarea returnată este un număr cuprins între `0` și `6`. Următoarea funcție poate fi folosită pentru a determina ziua din săptămână sub forma unui șir de caractere:

```
function convert(jour){
  semain.length=7;
  semain[0]="Dimanche";
  semain[1]="Lundi";
  semain[2]="Mardi";
  semain[3]="Mercredi";
  semain[4]="Jeudi";
  semain[5]="Vendredi";
  semain[6]="Samedi";
  return semain[jour];
}
```

Pentru a afișa ziua din săptămâna curentă poate fi folosită o secvență de tipul:

```
Aujourdhui=new Date();
document.writeln(convert(Aujourdhui.getDate());
```

Metoda getHours()

Metoda `getHours()` este folosită pentru a determina ora din cadrul datei stocate în obiectul pentru care este apelată; valoarea returnată este un număr cuprins între `0` și `23`. Sintaxa acestei metode este `date.getHours()`.

Metoda getMinutes()

Metoda `getMinutes()` este folosită pentru a determina minutul din cadrul datei stocate în obiectul pentru care este apelată; valoarea returnată este un număr cuprins între `0` și `59`. Sintaxa acestei metode este `date.getMinutes()`.

Metoda gerMonth()

Metoda `getMonth()` este folosită pentru a determina luna datei stocate în obiectul pentru care este apelată; valoarea returnată este un număr cuprins între `0` și `11`. Următoarea funcție poate fi folosită pentru a determina luna sub forma unui șir de caractere:

```
function convert(mois){
  année.length=12;
  année[0]="Janvier";
  année[1]="Février";
  année[2]="Mars";
  année[3]="Avril";
  année[4]="Mai";
  année[5]="Juin";
  année[6]="Juillet";
  année[7]="August";
  année[8]="Septembre";
  année[9]="Octobre";
  année[10]="Novembre";
  année[11]="Decembre";
  return année[mois];
}
```

Pentru a afișa luna curentă poate fi folosită o secvență de tipul:

```
Aujourdhui=new Date();
document.writeln(convert(Aujourdhui.getMonth()));
```

Metoda getSeconds()

Metoda `getSeconds()` este folosită pentru a determina secunda din cadrul datei stocate în obiectul pentru care este apelată; valoarea returnată este un număr cuprins între `0` și `59`. Sintaxa acestei metode este `date.getSeconds()`.

Metoda getTime()

Metoda `getTime()` este folosită pentru a determina numărul de milisecunde care au trecut începând din 1 Ianuarie 1970, ora 0:00. Sintaxa metodei este `date.getTime()`.

Chiar dacă un astfel de număr nu are o prea mare utilitate, funcția poate fi folosită pentru a determina intervalul de timp între două evenimente: se va apela funcția `getTime()` pentru obiectele dată corespunzătoare evenimentelor respective, iar intervalul de timp (în milisecunde) va fi indicat de diferența dintre cele două valori returnate.

Metoda getTimeZoneOffset()

Datorită faptului că în anumite aplicații trebuie să se lucreze cu un timp standard (independent de zona de fus orar în care se află vizitatorul) a fost introdusă funcția `getTimeZoneOffset()` care returnează diferența (în minute) dintre ora locală și ora GMT.

Metoda getYear()

Metoda `getYear()` este folosită pentru a determina anul din cadrul datei stocate în obiectul pentru care este apelată. Sintaxa acestei metode este

`date.getYears()`.

Metoda parse()

Metoda `parse()` este o metodă statică a clasei `Date`, adică o metodă care corespunde clasei și nu obiectelor instanță ale acesteia. Metoda are ca parametru un obiect de tip dată și returnează intervalul de timp (în milisecunde) dintre 1 Ianuarie 1970 și momentul indicat de obiectul dată respectiv. Metoda poate fi folosită într-o secvență de tipul:

```
date=new Date();
document.writeln(Date.parse(date));
```

Metoda setDate()

Metoda `setDate()` este metoda pereche pentru `getDate()`, ea permițând setarea zilei din lună pentru obiectul pentru care este apelată. Funcția are ca parametru un număr cuprins între `1` și `31`, sintaxa ei fiind

`date.setDate(jour)`.

Metoda setHours()

Metoda `setHours()` este metoda pereche pentru `getHours()`, ea permițând setarea orei pentru obiectul pentru care este apelată. Funcția are ca parametru un număr cuprins între `0` și `59`, sintaxa ei fiind

`date.setHours(heure)`.

Metoda setMinutes()

Metoda `setMinutes()` este metoda pereche pentru `getMinutes()`, ea permițând setarea minutului pentru obiectul pentru care este apelată. Funcția are ca parametru un număr cuprins între `0` și `59`, sintaxa ei fiind

`date.setDate(minute)`.

Metoda setMonth()

Metoda `setMonth()` este metoda pereche pentru `getMonth()`, ea permițând setarea lunii pentru obiectul pentru care este apelată. Funcția are ca parametru un număr cuprins între `0` și `11`, sintaxa ei fiind

`date.setMonth(mois)`.

Metoda setSeconds()

Metoda `setSeconds()` este metoda pereche pentru `getSeconds()`, ea permițând setarea secundei pentru obiectul pentru care este apelată. Funcția are ca parametru un număr cuprins între `0` și `59`, sintaxa ei fiind

`date.setSeconds(seconde)`.

Metoda setTime()

Metoda `setTime()` este metoda pereche pentru `getTime()`, ea permițând convertirea numărului de milisecunde cuprins în intervalul de timp 1 Ianuarie 1970, ora 0:00:00..ora curentă într-un obiect dată. Sintaxa acestei funcții este

```
date.setTime(milisecondes).
```

Metoda setYear()

Metoda `setYear()` este metoda pereche pentru `getYear()`, ea permițând setarea anului pentru obiectul pentru care este apelată. Sintaxa acestei funcții este

```
date.setYear(année).
```

Metoda toGMTString()

Metoda `toGMTString()` poate fi folosită pentru a converti o dată într-un șir care să aibă formatul standard GMT. Sintaxa funcției este `date.toGMTString()`, iar un exemplu de dată în formatul GMT standard este: Wed, 1 Dec 1999 12:45:07 GMT.

Metoda toLocaleString()

Metoda `toLocaleString()` poate fi folosită pentru a converti o dată într-un șir care să aibă formatul standard pentru fusul orar în care se află calculatorul utilizatorului. Sintaxa funcției este `date.toLocaleString()`, iar un exemplu de dată în formatul local standard este: 11/29/1999 23:33:55.

Metoda toString()

Valorile returnate de funcțiile prezentate anterior care furnizează valori sub forma unor șiruri de caractere nu sunt considerate string-uri veritabile în *JavaScript*, adică metodele specifice clasei `String` nu pot fi apelate pentru aceste valori. Pentru a putea apela aceste metode și pentru valorile de tip dată a fost introdusă metoda `toString()` care returnează data respectivă sub forma unui string. Sintaxa funcției este `date.toString()`.

Metoda UTC( )

Metoda `UTC()` are aproape același efect ca și metoda `parse()`, singura diferență fiind aceea că metoda `UTC()` lucrează cu timpul GMT, în timp ce metoda `parse()` lucrează cu timpul local. Există mai multe modalități de apelare a acestei funcții și anume:

```
Date.UTC(an, lună, zi);
Date.UTC(an, lună, zi, ore, minute, secunde);
Date.UTC(obiect_dată);
```

După cum se poate vedea și din modul de apelare, metoda `UTC()` este o metodă statică.

Ferestre în JavaScript

În *JavaScript* o fereastră este considerată a fi o reprezentare "fizică" a ceea ce se întâmplă în "interiorul" programului de navigare. Până acum am folosit numai zona *client* a ferestrei programului de navigare, dar limbajul *JavaScript* oferă și posibilitatea creării și controlării comportamentului ferestrelor prin intermediul unor funcții predefinite.

Ferestre predefinite (de dialog)

Limbajul *JavaScript* pune la dispoziție trei funcții pentru crearea de ferestre de dialog: `alert()`, `confirm()` și `prompt()`. Cu toate că nu avem controlul asupra locului în care este afișată fereastra, asupra tipului butoanelor din fereastră sau asupra dimensiunilor ferestrei, aceste ferestre oferă un mecanism de informare a utilizatorului asupra anumitor evenimente; de asemenea este facilitată preluarea datelor de la utilizator.

Funcția alert()

Prin apelul funcției `alert()` este creată o fereastră în interiorul căreia sunt afișate anumite informații destinate utilizatorului. Sintaxa acestei funcții este: `alert(mesaj)`.

Fereastra creată are un singur buton: **`Ok`**, nu poate fi minimizată sau redimensionată, singura posibilitate de a "scăpa" de ea fiind apăsarea butonului **`Ok`**. Până la apăsarea acestui buton programul de navigare nu mai efectuează nici o operație; din această cauză s-ar putea să apară efecte inestetice cum ar fi încărcarea parțială a unor pagini.

În figura 1 poate fi observată o fereastră de avertizare specifică sistemului de operare *Microsoft Windows 98 Second Edition* care a fost generată folosind un apel al funcției `alert()`.

Fig. 1. Fereastră de avertizare

Funcția confirm()

Prin apelul funcției `confirm()` este creată o fereastră prin intermediul căreia se cere confirmarea utilizatorului pentru efectuarea unei acțiuni. Funcția returnează o valoare logică (`true` sau `false`), iar sintaxa ei este

```
boolvalue=confirm(mesaj).
```

Fereastra creată are două butoane: **`Ok`** și **`Cancel`**, nu poate fi minimizată sau redimensionată și are un buton special pentru închidere. În cazul apăsării butonului **`Ok`** sau a tastei `ENTER` valoarea returnată este `true`, iar în cazul apăsării butonului **`Cancel`**, a butonului de închidere sau a tastei `ESC` valoarea returnată este `false`.

În figura 2 este prezentată o fereastră de confirmare specifică sistemului de operare *Microsoft Windows 98 Second Edition* care a fost generată folosind un apel al funcției `confirm()`.

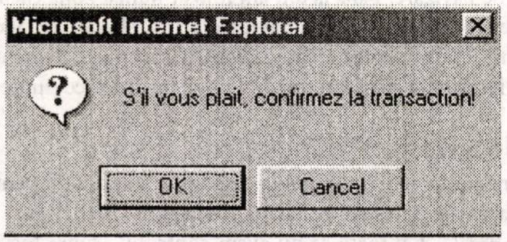


Fig. 2. Fereastră de confirmare

Funcția prompt()

Prin apelul funcției `prompt()` va fi creată o fereastră care va permite utilizatorului să introducă un șir de caractere. Funcția va returna șirul de caractere introdus sau valoarea `"null"`, iar sintaxa ei este

`stringvalue=prompt(`*mesage[,valoare_implicită]*`).`

Fereastra creată conține un mesaj indicat prin intermediul parametrului `mesage`, o casetă de tip text în interiorul căreia utilizatorul va putea introduce un șir de caractere, are două butoane: **`Ok`** și **`Cancel`**, nu poate fi minimizată sau redimensionată și are un buton special pentru închidere. În cazul în care apare parametrul `valoare_implicită`, în momentul apariției ferestrei caseta de tip text destinată introducerii datelor de către utilizator va conține valoarea acestui parametru. În cazul apăsării butonului **`Ok`** sau a tastei `ENTER` funcția va returna un string care va conține textul din casetă, iar în cazul apăsării butonului **`Cancel`**, a butonului de închidere sau a tastei `ESC` valoarea returnată este `"null"`.

În figura 3 este prezentată o fereastră pentru introducerea datelor specifică sistemului de operare *Microsoft Windows 98 Second Edition* care a fost generată folosind un apel al funcției `prompt()`.

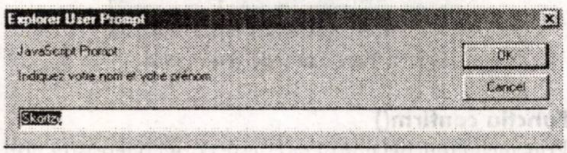


Fig. 3. Fereastră pentru introducerea datelor

Ferestre ale programului de navigare

Limbajul *JavaScript* oferă posibilitatea deschiderii de noi ferestre ale programului de navigare care pot fi controlate prin intermediul acestui limbaj.

Funcția open()

Această funcție poate fi folosită pentru a crea o nouă fereastră; într-o oarecare măsură ea este echivalentă cu deschiderea unei noi ferestre în interiorul programului de navigare folosind comenzi specifice acestuia:

- pentru *Internet Explorer 5.0* se alege **`File, New, Window`** sau se apasă simultan tastele `CTRL` și `N`.
- pentru *Netscape Navigator 4.7* se alege **`File, New, Navigator Window`** sau se apasă simultan tastele `CTRL` și `N`.

Programatorul va putea controla felul în care este afișată această fereastră precum și conținutul ei. Sintaxa funcției `open()` este:

`open(URL,nume_fereastră[,atribute_fereastră]).`

Parametrul `URL` este un șir de caractere care indică URL-ul paginii Web care va fi încărcată în interiorul noii ferestre. Parametrul `nume_fereastră` este tot un șir de caractere care va indica denumirea noii ferestre; aceasta va fi denumirea folosită pentru atributul **`target`** al marcajelor **`<FORM>`** și **`<A>`** și este echivalentă cu valoarea atributului **`name`** al acestor marcaje. Prin intermediul parametrului `atribute_fereastră` pot fi setate diferite caracteristici ale noii ferestre; o listă cu o parte dintre caracteristicile unei ferestre sunt prezentate în tabelul 1.

Caracteristicile sunt indicate prin perechi de forma `caracteristică=valoare`; în cazul caracteristicilor `toolbar`, `location`, `status`, `menubar`, `scrollbars` și `resizable`, `valoare` poate lua valorile `yes` (sau `1`) și `no` (sau `0`); în cazul caracteristicilor `width` și `height`, `valoare` poate avea ca valori numere naturale.

În cazul în care nu este specificată nici o valoare pentru nici o caracteristică, se consideră că toate caracteristicile au valoarea `yes`.

În cazul în care este indicată o valoare pentru cel puțin una dintre caracteristici, pentru toate celelalte caracteristici pentru care nu au fost specificate valori se va considera că valoarea este `no`. Dacă nu sunt indicate valori pentru caracteristicile `height` și `width`, noua fereastră va

Tabelul 1 Caracteristici ale ferestrelor

Caracteristică	Descriere
toolbar	Controlează afișarea barei de unelte
location	Controlează afișarea câmpului care conține locația curentă
status	Controlează afișarea barei de stare
menubars	Controlează afișarea barei de meniuri
scrollbars	Controlează afișarea barelor de defilare
resizable	Indică faptul dacă utilizatorul poate redimensiona fereastra
width	Indică lățimea ferestrei (în pixeli)
height	Indică înălțimea ferestrei (în pixeli)

ocupa întreg ecranul. Un exemplu de utilizare a funcției `open()` este:

```
NewWindow=open("www.ginfo.ro","GInfo",
"menubar=yes,location=no,status=yes,
height=200,width=500").
```

După cum se poate vedea și din exemplul anterior, în cazul în care sunt specificate valori pentru mai multe caracteristici, perechile `caracteristică=valoare` vor fi separate prin virgule.

Funcția close()

Funcția `close()` este complementara funcției `open()`, ea putând fi folosită pentru închiderea unei ferestre. Sintaxa acestei funcții este `fereastră.close()`; de exemplu, pentru a închide fereastra curentă poate fi folosit oricare din următoarele tipuri de apeluri:

- `window.close()`
- `self.close()`
- `close()`

Pentru a închide o altă fereastră, funcția `close()` trebuie apelată pentru obiectul corespunzător ferestrei respective. De exemplu, pentru a închide fereastra *GInfo* deschisă prin intermediul secvenței prezentate anterior, va trebui să scriem: `NewWindow.close()`.

Obiecte fereastră

După cum reiese și din cele prezentate anterior, în *JavaScript*, fiecărei ferestre îi este asociat un obiect. Cele mai importante proprietăți ale unui astfel de obiect sunt caracteristicile ferestrei: `toolbar`, `location`, `status`, `menubar`, `scrollbars`, `resizable`, `width`, `height`. Prin intermediul acestor proprietăți pot fi modificate caracteristicile unei ferestre prin apeluri de forma:

`fereastră.proprietate=valoare.`

Pentru a crea fereastra GInfo prezentată anterior am putea folosi și o secvență de tipul:

```
NewWindow=open("www.ginfo.ro","GInfo");
NewWindow.menubar="yes";
NewWindow.location="no";
NewWindow.status=1;
NewWindow.height=200;
NewWindow.width=500;
```

Cea mai importantă și mai des folosită metodă a obiectelor fereastră este `close()`.

Mihai Scorțaru este student în anul II la Universitatea Tehnică Cluj. Poate fi contactat prin e-mail la smihai@asterix.obs.utcluj.ro

news...

A apărut Internet Explorer 5.01

Microsoft a anunțat că noua versiune Internet Explorer 5.01 este disponibilă pentru download gratuit pe site-ul propriu. Internet Explorer 5.01 poate rula sub Windows 95, Windows 98 și Windows NT 4.0 și conține o serie de îmbunătățiri în ceea ce privește sistemul de securitate și alte câteva îmbunătățiri minore. Dintre acestea se pot aminti un update la tehnologia de criptare pe 56-de biți sau suportul pentru S/MIME versiunea 3.

Netscape vs. Microsoft

Netscape Communications va lansa în curând o versiune beta a pachetului de programe Netscape Communicator 5.0, dar va rămâne în continuare în urmă față de Microsoft Internet Explorer. Această versiune va apărea cu aproape un an întârziere față de data prevăzută inițial. Cu toate acestea, Netscape Communicator 5.0 nu va reuși să se ridice la nivelul programului de navigare Microsoft Internet Explorer 5.0 în ceea ce privește suportul pentru tehnologiile Web avansate, cum ar fi XML (Extensible Markup Language). Acest lucru se simte și în preferința pieței pentru Microsoft Internet Explorer față de Netscape Communicator.

Oficialii Netscape Communications declară însă că merită așteptată lansarea noului produs, care este considerat revoluționar. Dar toate noile caracteristici pe care le promite Netscape Communicator 5.0 sunt oferite de Microsoft Internet Explorer 5.0 de aproape un an, deci nimic nu mai este revoluționar.

În concluzie s-ar putea spune că Netscape Communications a pierdut "războiul browser-elor".

NORTON CleanSweep
NORTON Utilities
NORTON AntiVirus
EasyRestore
PartitionMagic
LOST & FOUND
ROMSYM DATA-SRL
SOFTWARE
BRINEL
ALPIS
ALPIS®
Designed for Microsoft Windows NT Windows 98
WINDOWS 2000 READY PCs
ISO-9002
COMPAQ
P/N : 131015-a82
Deskpro EN SFF C466
6.4 GB Hard
64 MB RAM
NIC / Sound / W95
Monitor 15"
MAINBOARD
QDI WinneX 1, Intel 82810,
Intel Celeron 400PPGA W/128K cache
32 MB SDRAM DIMM PC 100 (8ns)
SVGA OB BOARD 4MB AGP
HDD 6.4 Quantum Fireball
CD-ROM 40x IDE
SOUND CARD on board +
Stereo Speakers 60W PMPO
FLOPPY 3.5", 1.44MB
ACER PS2 Keyboard WIN'95 +
Mouse SERIAL 3b + Pad
MONITOR COLOR
SVGA SMILE 14" DIGITAL
COREL DRAW 8
OMNILOGIC BGS
SALIENT
SALIENT Romania
MULTIMEDIA KIT
• CD-ROM CREATIVE LABS
• SOUND CARD 16bit
• BOXE 80W
SPRINT
MONITOR SPRINT 17" - model MP 1770N
PixelView
PixelView PlayTVpro
PV-BT878P+(W/FM)
Watch TV +Play TV+ • Motion Capture+
Teletext+ • Closed Capture+
FM Radio + • Remote Control+
Video Camera+ • Video Conferencing Ready
oferit de
ACER
THE COMPUTER COMPANY
Anul 2000 se lasă cu
PREMII...
COMPUTER PRESS AGORA

Microsoft România

Microsoft Office 2000 Professional

TORNADO SISTEMS

Canon

Canon Bubble Jet
Color Printer BJC-4400

Iomega Zip Drive
250 MB

GT Grup Transilvae

Procesor: AMD-K6 400 MHz 3DNow!
64 MB RAM • HD 6,7 Gb
Video: S3 Savage3D 8 MB Monitor 17"
Sound card pe placă de bază: EESS TM ES1938S Solo-1
CD-ROM: LITEON LTN382 40X
Tastatură: Qtronix multimedia Scorpius-98N
Mouse: Qtronix Lynx 3D
Modem: HSF 56k date/fax/voce intern, cu cipset Rockwell
Boxe: Mecmar LS-26A, 135W PMPO

UMAX Astra 1220U
rezoluție optică 600x1200dpi
max. 9600x9600, col. 36bit, 1 pass

HEWLETT PACKARD
ROMÂNIA

HP DeskJet 610 C

Placa de bază ATX, INTEL 810, W/4 MB AGP&SOUND, UDMA 66
Procesor INTEL CELERON 400 (128ke) PPGA • Memorie 32 MB SDRAM, PC-10
Video 4 MB AGP ON BOARD • Hard-disk QUANTUM 6.4 GB UDM
Tastatură GENIUS WIN 96 • Floppy 1.44 MB FDD • Mous SCROLLI
Monitor 14" SVGA LR NI DIGITAL MPRII • CD-ROM
Sound ON BOARD AC 97 AUDIO
Boxe 120 W, ACTIVE SPEAK S

ABONEAZĂ-TE și câștigă!

Abonează-te
la PCMagazine România,
PC Report sau Gazeta de informatică,
reviste editate de Computer Press Agora, până la data de 15 decembrie 1999.
Vei participa la tragerea la sorți care va desemna câștigătorii premiilor.
Trimite cărțile poștale din aceste reviste pe adresa:
Computer Press Agora, str.T.Vladimirescu nr.63/1, Tg.Mureș 4300 (CP 230/1)
Pentru informații suplimentare poți suna la: tel. 065-166516, fax 065-166290

GEOMETRIE computațională

Angel Proorocu

Ne propunem să prezentăm un algoritm de determinare a intersecției a două poligoane convexe. Această problemă a fost tratată în Gînfo nr 4-5/1998, dar algoritmul prezentat atunci avea complexitatea O(n*m). Vom prezenta acum un algoritm de complexitate liniară, respectiv de O(n+m).

Problemă

Fie în planul **P** două poligoane convexe `P` și `Q`, cu `n` și respectiv `m` puncte. Fiind date aceste poligoane, să se găsească intersecția lor.

Algoritmul pe care îl vom prezenta este util în rezolvarea multor probleme, ca de exemplu în determinarea faptului că două mulțimi de puncte sunt separabile printr-o dreaptă. Prima rezolvare liniară a fost găsită de *Shamos* (1978), dar noi vom prezenta algoritmul introdus în 1982 de *O'Rouke*, *Chien*, *Olson* și *Naddor*. Ideea de bază a algoritmului este simplă, dar implementarea este delicată, cum se întâmplă de obicei la problemele de geometrie analitică. Înainte de prezentarea algoritmului, trebuie să subliniem un aspect evident dar foarte important: intersecția a două poligoane convexe este vidă sau este un poligon convex.

Presupunând că punctele poligoanelor sunt date în ordinea inversă acelor de ceasornic nu restrângem generalitatea algoritmului.

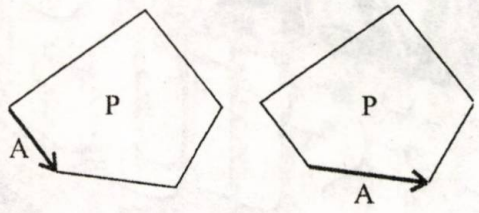


Fig. 1. Poligonul P și deplasarea vectorului A

Fie `A` și `B` doi vectori care se suprapun, `A` pe una din laturile lui `P` și `B` pe una din laturile lui `Q`, ca în figura 1. În algoritm, vectorii `A` și `B` se deplasează pe conturul poligoanelor `P` și `Q` în sens trigonometric astfel încât se suprapun în fiecare punct de intersecție al poligoanelor. La început, vectorii sunt suprapuși pe orice latură. În continuare trebuie să decidem la fiecare pas care vector trebuie deplasat.

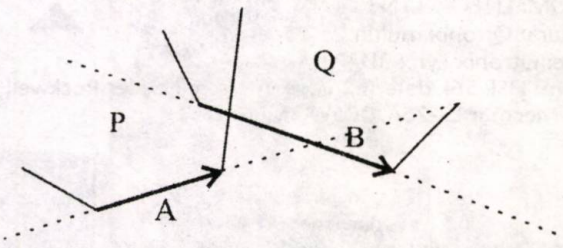


Fig. 2. Putem scrie A→B, dar nu putem scrie B→A, deoarece A indică spre B

Notăm cu `A→B` faptul că prelungirea vectorului `A` în sensul de deplasare întâlnește dreapta suport a lui `B`; notăm analog cu `B→A` faptul că prelungirea vectorului `B` în sensul de deplasare întâlnește dreapta suport a lui `A` (figura 2). Mai notăm cu `H(A)` semiplanul din stânga lui `A`, `H(B)` semiplanul din stânga lui `B`, `b` punctul din "vârful" lui `B` și `a` punctul din "vârful" lui `A`.

Cu notațiile descrise, condițiile de deplasare se pot scrie astfel:

1) `A→B` ⇒ deplasăm vectorul A
2) `B→A` ⇒ deplasăm vectorul B
3) `A→B` și `B→A` ⇒ deplasăm oricare dintre vectorii `A` și `B`
4) nu este îndeplinită nici una din condițiile `A→B` și `B→A` ⇒ dacă `b∈H(A)` atunci avansăm `A`, altfel avansăm `B`.

Algoritmul în pseudocod este următorul:

```
Alege A și B oarecare
inside ∈ {unknown,Pin,Qin}
        { arată dacă un poligon este inclus în celălalt }
repetă
  dacă A se intersectează cu B
  atunci se modifică variabila inside;
         punctul de intersecție este adăugat poligonului
         soluție;
```

se avansează A *sau* B *în funcție de condițiile de mai sus;*
punctul din "coada" vectorului avansat este adăugat poligonului soluție dacă variabila inside *arată că poligonul pe care se deplasează vectorul avansat este în interior;*

până când nr_iterații>2(n+m);

Se tratează particular cazurile: P∩Q=Ø, P⊆Q, Q⊆P.

Notând cu AxB produsul vectorial al lui A și B, condițiile geometrice de deplasare pot fi scrise astfel în funcție de semnul lui AxB și de valoarea de adevăr a propozițiilor a∈H(B) și b∈H(A):

AxB	a∈H(B)	b∈H(A)	Vectorul deplasat
>0	T	T	A
>0	T	F	A sau B
>0	F	T	A
>0	F	F	B
<0	T	T	B
<0	T	F	B
<0	F	T	A sau B
<0	F	F	A

Echivalența între aceste condiții și cele expuse mai sus s-ar putea să nu fie foarte clară la prima vedere, însă spațiul nu ne permite o demonstrație riguroasă. Algoritmul este însă intuitiv și, după rularea câtorva exemple, vă veți convinge de acuratețea formulelor.

Programul Pascal, care realizează vizualizarea grafică a algoritmului, este prezentat în continuare. Lăsăm pe seama cititorului tratarea cazurilor speciale P∩Q=Ø, P⊆Q, Q⊆P.

Programul citește din fișierul **INPUT.TXT** două poligoane în următorul format:

- prima linie conține numărul de puncte n ale primului poligon;
- următoarele n linii conțin perechi de numere reale X_i, Y_i (i=1..n) reprezentând coordonatele punctelor poligonului în ordine trigonometrică;
- a n+2-a linie conține numărul de puncte m pentru al doilea poligon;
- următoarele M linii conțin perechi de coordonate reale X_i, Y_i (i=1..n) reprezentând coordonatele punctelor poligonului în ordine trigonometrică.

Programul afișează pe ecran cele două poligoane, apoi vizualizează pașii algoritmului. Poligoanele sunt desenate cu verde, respectiv roșu, iar vectorii A și B cu alb, respectiv galben. La fiecare apăsare a tastei ENTER, se avansează cu unul dintre cei doi vectori, iar după terminarea algoritmului se vizualizează intersecția celor două poligoane. Funcțiile și procedurile programului sunt explicate în comentarii.

Listing: INTER.PAS

```
{$M 65384,0,655360}
Program IntersectiePoligoane;
uses Graph,Crt;
type TPoint=array[0..1]of Real;
{ contine coordonatele x, y ale unui punct }
     Tinflag=(Pin,Qin,unknown);
      { ne indica daca unul din poligoane }
      {      este in interiorul celuilalt }
     TSegment=array[0..1]of TPoint;
      {     un segment-contine doua puncte }
     TIntPoint=array[0..1]of Integer;
      {     un punct cu coordonate intregi }
      {              pentru afisarea pe ecran }
     TPolyPoints=array[0..1000]of TIntPoint;
      {      poligon cu coordonate intregi }
      {             pentru afisarea pe ecran }
var n,m:Integer;
   { nr de puncte ale poligoanelor P si Q }
    PolyPoints:TPolyPoints;
    minx,maxx,maxy,miny:Real;
    P,Q,Sol:array[0..1000]of TPoint;
                                { poligoanele }
    f:Text;
    inflag:Tinflag;
                { indicatorul ce poate fi }
                {    Pin, Qin sau unknown }
    gd,gm,na,mb,a,b,a1,b1,ns:Integer;
    AB,BA:Boolean;

procedure Linie(x,y,xx,yy:Real);
            { deseneaza o linie pe ecran }
begin
    { se convertesc coordonatele reale la }
    {     coordonate ecran, in functie de }
    {        minimul/maximul pe x si pe y }
  x:=639*(1-(maxx-x)/(maxx-minx));
  xx:=639*(1-(maxx-xx)/(maxx-minx));
  y:=479*(1-(maxy-y)/(maxy-miny));
  yy:=479*(1-(maxy-yy)/(maxy-miny));
  Line(Trunc(x),479-Trunc(y),Trunc(xx),
                                479-Trunc(yy))
end;

procedure ReadData;
{ citeste datele,initializeaza modul grafic }
var i,j,grDriver,ErrCode,grMode:Integer;
begin
  Assign(f,'INPUT.TXT'); Reset(f);
  Readln(f,n);
  for i:=0 to n-1 do Readln(f,P[i][0],P[i][1]);
  minx:=Maxlongint; maxx:=-Maxlongint;
  miny:=Maxlongint; maxy:=-Maxlongint;
{   se determina minimul si maximul coord. }
{ punctelor din poligoane pentru functia de }
{    convertire de la real la coord. ecran }
```

```
  for i:=0 to n-1 do
  begin
    if p[i][0]<minx then minx:=p[i][0];
    if p[i][0]>maxx then maxx:=p[i][0];
    if p[i][1]<miny then miny:=p[i][1];
    if p[i][1]>maxy then maxy:=p[i][1]
  end;
  Readln(f,m);
  for i:=0 to m-1 do Read(f,Q[i,0],Q[i,1]);
  for i:=0 to m-1 do
  begin
    if q[i][0]<minx then minx:=q[i][0];
    if q[i][0]>maxx then maxx:=q[i][0];
    if q[i][1]<miny then miny:=q[i][1];
    if q[i][1]>maxy then maxy:=q[i][1]
  end;
  Close(f); a1:=0; b1:=0;
  a:=(a1+1) mod n; b:=(b1+1) mod n;
  inflag:=unknown;
     { la inceput nu se stie care poligon }
     {   este in interiorul celuilalt      }
  na:=0; mb:=0; grDriver:=Detect;
  InitGraph(grDriver, grMode,'c:\bp\bgi');
  SetColor(green);
  for i:=0 to n-1 do
  begin
    j:=(i+1)mod n;
    Linie(p[i][0],p[i][1],p[j][0],p[j][1])
  end;
  SetColor(red);
  for i:=0 to m-1 do
  begin
    j:=(i+1)mod m;
    Linie(q[i][0],q[i][1],q[j][0],q[j][1])
  end
end;

function Arie(a,b,c:Tpoint):Real;
{ calculeaza aria*2 a unui triunghi,semnul }
{    functiei depinde de ordinea punctelor: }
{ exista o echivalenta intre semnul acestei }
{        functii si produsul vectorial AxB }
begin
  Arie:=a[0]*b[1]-a[1]*b[0]+b[0]*c[1]-
              c[0]*b[1]+c[0]*a[1]-a[0]*c[1];
end;

procedure WriteSol;
var i:Integer;
begin
  Assign(f,'OUTPUT.TXT');
  Rewrite(f);
  for i:=1 to ns do Writeln(f,sol[i][0]:0:3,
                        ' ',sol[i][1]:0:3);
  Close(f)
end;
```

```
function Stanga(A:TSegment;
pB:TPoint):Boolean;
begin { returneaza true numai daca punctul }
      {    pB este in stanga segmentului A }
  Stanga:=(Arie(A[0],A[1],pB)>0)
end;

function Intersect(A,B:TSegment;
                   var P:TPoint):Boolean;
var s,t,det:Real;
begin
{ calculeaza intersectia segmentelor A si B }
{  returneaza true numai daca segmentele se }
{   intersecteaza; nu vom detalia procedura }
{       deoarece o metoda similara a facut  }
{ subiectul  unui articol anterior in GInfo }
  det:=A[0][0]*(B[1][1]-B[0][1])+
       A[1][0]*(B[0][1]-B[1][1])+
       B[1][0]*(A[1][1]-A[0][1])+
       B[0][0]*(A[0][1]-A[1][1]);
  if det=0
  then begin Intersect:=false; Exit end;
  s:=(A[0][0]*(B[1][1]-B[0][1])+
      B[0][0]*(A[0][1]-B[1][1])+
      B[1][0]*(B[0][1]-A[0][1]))/det;
  t:=-(A[0][0]*(B[0][1]-A[1][1])+
       A[1][0]*(A[0][1]-B[0][1])+
       B[0][0]*(A[1][1]-A[0][1]))/det;
  P[0]:=A[0][0]+S*(A[1][0]-A[0][0]);
  P[1]:=A[0][1]+S*(A[1][1]-A[0][1]);
  Intersect:=(0.0<=s)and(s<=1.0)and(0.0<=t)
                                  and(t<=1)
end;

function SgnArieOrg(A,B:TSegment):Integer;
var p1,p2,p3:TPoint;    aux:Real;
begin
{ calculeaza semnul produsului vectorial AxB }
  p1[0]:=0; p1[1]:=0;
  p2[0]:=A[1][0]-A[0][0];
  p2[1]:=A[1][1]-A[0][1];
  p3[0]:=B[1][0]-B[0][0];
  p3[1]:=B[1][1]-B[0][1];
  aux:=Arie(p1,p2,p3);
  if aux=0 then SgnArieOrg:=0
  else
    if aux>0 then SgnArieOrg:=1
    else SgnArieOrg:=-1
end;

procedure Avanseaza(var c,a,a1,n:Integer;
                    inside:Boolean; P:TPoint);
begin
     { avanseaza vectorul de pe poligonul }
     {           precizat prin parametri  }
  Inc(c);
```

```
  if inside
  then begin Inc(ns); sol[ns]:=P end;
  a1:=a; a:=(a+1) mod n
end;

procedure Rezolva;
var pP:TPoint;
    AA,BB:TSegment;
    cross_over:Boolean;
    i,Sign:Integer;
begin
   { numarul de puncte ale polig. solutie }
  ns:=-1;
  repeat   { AA si BB sunt vectorii ce se }
           { ataseaza poligoanelor }
    AA[0]:=P[a1]; AA[1]:=P[a];
    Setcolor(white);
    linie(aa[0,0],aa[0,1],aa[1,0],aa[1,1]);
    BB[0]:=Q[b1]; BB[1]:=Q[b];
    Setcolor(yellow);
    linie(bb[0,0],bb[0,1],bb[1,0],bb[1,1]);
    Readln;
    AB:=Stanga(AA,BB[1]); BA:=Stanga(BB,AA[1]);
    cross_over:=Intersect(AA,BB,pP);
    Sign:=SgnArieOrg(AA,BB);
    if cross_over
    then { daca vectorii se intersecteaza }
    begin
      if inflag=unknown
      then begin na:=0; mb:=0 end;
      Inc(ns); Sol[ns]:=pP;
      if BA
      then inflag:=Pin
      else inflag:=Qin
{  se seteaza flagul ce arata care poligon }
{   este inautru, interior fiind poligonul }
{ care este in stanga celuilalt, deoarece }
{sensul de parcurgere este cel trigonometric }
    end;
{  conditiile multiple ce urmeaza nu sunt }
{ decat implementarea cazurilor expuse in }
{ articol, in functie de AxB si de varfurile }
{ vectorilor variabila sign contine semnul }
{ lui AxB, iar in functie de valorile lui }
{ BA si AB se stabilesc celelalte conditii }
    if (Sign=0) and not AB and not BA
    then
    begin
    if inflag=Pin
    then Avanseaza(mb,b,b1,m,inflag=Qin,Q[b])
{       la apelarea functiei avanseaza s-a }
{ folosit un mic artificiu de programare, }
{   pentru a putea scrie functia cat mai }
{   compact, fara conditii pentru fiecare }
{ poligon in parte, diferind doar modul de }
{          apleare, in functie de poligon }
    else Avanseaza(na,a,a1,n,inflag=Pin,P[a])
    end
    else
      if Sign>=0
      then
      begin
        if AB then Avanseaza(na,a,a1,n,
                             inflag=Pin,P[a])
             else Avanseaza(mb,b,b1,m,
                             inflag=Qin,Q[b])
      end
    else
    begin
      if BA then Avanseaza(mb,b,b1,m,
                             inflag=Qin,Q[b])
      else Avanseaza(na,a,a1,n,inflag=Pin,
                                   P[a])
    end;
  Setcolor(green);
  linie(aa[0][0],aa[0][1],aa[1][0],aa[1][1]);
  Setcolor(red);

linie(bb[0][0],bb[0][1],bb[1][0],bb[1][1]);
          { se reprezinta grafic pasul facut }
  until (na>2*n)or(mb>2*m)or((mb>m)and(na>n));
  writeSol;          { se afiseaza solutia }
  for i:=0 to ns do
  begin
    sol[i][0]:=639*(1-(maxx-sol[i][0])/
                                 (maxx-minx));
    sol[i][1]:=479*(1-(maxy-sol[i][1])/
                                 (maxy-miny));
    polypoints[i][0]:=Trunc(sol[i][0]);
    polypoints[i][1]:=479-Trunc(sol[i][1])
  end;
{  se convertesc coordonatele poligonului }
{             solutie la coordonate ecran }
    Fillpoly(ns,PolyPoints);
{ se vizualizeaza intersectia poligoanelor }
    Readln;
    Closegraph { inchiderea modului grafic }
end;

Begin
  ReadData; Rezolva
End.
```

Bibliografia:

1. **Joseph O. Rourke** - *Computational Geometry in C*, Cambridge University Press 1993
2. **Ion D. Teodorescu** - *Geometrie analitică și elemente de algebră liniară*, Editura didactică și pedagogică București, 1972

Angel Proorocu este elev în clasa a XII-a la Liceul de Informatică din București. Poate fi contactat prin e-mail la psangel@pcnet.pcnet.ro

Un MODEL pentru problema ORARULUI

Vlad Bazon

Problema orarului este una dintre cele mai cunoscute probleme de algoritmică, până în prezent necunoscându-se nici un algoritm eficient pentru rezolvarea ei.

Formulare generală

Se dă matricea încadrărilor - precizând pentru fiecare profesor numărul de ore la fiecare clasă. Cerința principală este aceea de a obține un orar: o repartizare a profesorilor la clase, pe zilele săptămânii și în cadrul fiecărei zile, în așa fel încât într-o aceeași oră (din zi):

- fiecare profesor să intre la cel mult una dintre clasele asociate lui prin matricea de încadrare;
- la fiecare clasă să intre cel mult câte unul dintre profesorii asociați clasei.

Particularizări remarcabile

În general, judecarea unor cazuri particulare este utilă pentru înțelegerea problemei, a nuanțelor și limitelor ei și pentru intuirea complexității și a șanselor de aplicare practică a unei metode sau alta; în sfârșit, particularizările respective vor fi importante pentru faza de testare și punere la punct a programului.

Pătrate latine

5	9	11	7	1	2	6	10	8	4	3	
9	5	7	11	2	1	4	6	10	3	8	
11	7	5	9	10	3	8	4	1	6	2	
7	11	9	5	3	10	2	8	6	1	4	
1	6	4	2	5	9	10	3	7	8	11	
6	1	8	10	9	5	3	2	4	11	7	
4	8	1	3	11	7	5	9	2	10	6	
8	4	6	1	7	11	9	5	3	2	10	
2	3	10	6	8	4	11	1	5	7	9	
10	2	3	4	6	8	7	11	9	5	1	
3	10	2	8	4	6	1	7	11	9	5	

Fig. 1 - pătrat latin de ordinul 11

Presupunem că matricea de încadrare este pătratică și are toate elementele egale cu 1 (adică numărul de profesori este egal cu numărul de clase și fiecare profesor are câte o singură oră la fiecare clasă).

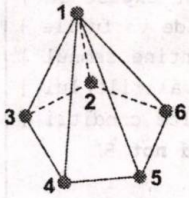
Atunci, repartizarea în cadrul unei singure "zile" a orelor respective conduce la un ***pătrat latin***: pe fiecare linie ("oră" a zilei) și de asemenea, pe fiecare coloană ("clasă"), apar toți profesorii (câte o singură dată).

Orarul unui turneu de șah

Presupunem o matrice de încadrare pătratică, în care valorile de pe diagonala principală sunt egale cu 0 și toate celelalte elemente au valoarea 1.

Presupunem că orarul cerut trebuie să îndeplinească următoarea condiție: pentru oricare numere de ordine *J* și *K*, *dacă profesorul J are oră la clasa K, atunci simultan, profesorul K are oră la clasa J* (cu altă interpretare: dacă *J* "joacă" cu *K*, atunci și *K* joacă cu *J*).

Identificând "profesorii" și "clasele" cu participanții la un turneu de șah (unde fiecare trebuie să joace câte o singură partidă cu fiecare dintre ceilalți jucători), se obține un orar de desfășurare pe runde ("zile"), a partidelor.



runda	partidele
I	3-4 2-5 1-6
II	4-5 3-6 1-2
III	6-5 2-4 1-3
IV	2-6 3-5 1-4
V	2-3 6-4 1-5

	clasele					
ora	1	2	3	4	5	6
I	6	5	4	3	2	1
II	2	1	6	5	4	3
III	3	4	1	2	6	5
IV	4	6	5	1	3	2
V	5	3	2	6	1	4
	profesorii					

piramida jucătorilor *rundele turneului* *orar-turneu*

Fig. 2 - Obținerea rundelor unui turneu de șah

Pentru acest caz există și un algoritm foarte simplu (ca formulare practică) pentru obținerea orarului ([1]). Acest algoritm se poate deduce urmărind Fig. 2: o rundă se obține selectând întâi *o latură* (neselectată anterior) a bazei piramidei (care este poligon regulat cu număr impar de

vârfuri), adăugând apoi *diagonalele* succesive ale bazei care sunt paralele cu latura selectată și completând cu *muchia* laterală corespunzătoare vârfului rămas bazei (se poate presupune că există un număr par de jucători - implicând în caz contrar, un jucător fictiv).

Termenii de bază ai problemei

Datele inițiale sunt numerele

`nq`=numărul de clase (numărul de coloane ale matricei încadrărilor),

`nZ`=numărul de zile,

`nO`=numărul (maxim) de ore / zi,

`nP`=numărul de profesori (numărul de linii ale matricei încadrărilor).

și matricea încadrărilor. Numele profesorilor, denumirile claselor și alte asemenea date inițiale au importanță numai pentru formularea finală a orarului.

Vom nota apoi:

`nS=nZ*nO` - numărul (maxim) de ore / săptămână la clase,

`nV=nq*nS` - numărul total de ore (inclusiv toate orele libere), cu precizările următoare (modelând unele condiții firești, întâlnite în practică): există măcar o clasă căreia matricea încadrărilor îi asociază (în total) exact `nS` ore (dar niciodată mai mult de `nS` ore); oricare profesor `1..nP` are cel mult `nS` ore în încadrarea sa; matricea încadrărilor ar trebui să asocieze fiecărei clase cel puțin `nS-(nZ-1)` ore; altfel spus, la fiecare clasă pot exista cel mult `(nZ-1)` ore libere ("în plus" față de cele atribuite efectiv, prin matricea încadrărilor).

Toate orele libere (dacă există) sunt atribuite unui profesor *fictiv* (cu numărul de ordine `0`); orele libere la clase distincte se pot desfășura simultan (într-o aceeași oră din cadrul unei zile).

Termenul "oră"

În general, "oră" are mai multe înțelesuri; în contextul nostru sunt utile următoarele două semnificații:

- "oră" poate desemna o *asociere posibilă* (bazată pe matricea încadrărilor) între o anumită clasă și un anumit profesor (eventual, cel fictiv);
- "oră" poate avea și o semnificație temporală - implicând *simultan* toate clasele (și un anumit număr de profesori).

Vom numi ***S-oră*** orice secvență de `nq` ore (asocieri posibile profesori-clase, cu respectarea ordinii claselor), care sunt posibile simultan (se pot desfășura într-o aceeași oră din cadrul unei aceleiași zile).

Astfel, orarul unei zile este constituit din `nO` *S*-ore (și facem abstracție de ordinea acestora); în total, pe întreaga săptămână - avem de stabilit un număr de `nS` *S*-ore, astfel încât ansamblul acestora să fie compatibil cu matricea încadrărilor (și cu eventualele condiționări suplimentare).

2.1 Trei principii de bază pentru rezolvare

Ne bazăm soluționarea pe următoarele trei idei:

1. ***Numerotarea orelor***, astfel încât să se poată recunoaște (ușor) asocierea profesor-clasă aferentă. În principiu, numerotăm cele `nV` ore modulo (`nV`) prin `0, 1, 2,..., nV-1`; numerotăm clasele modulo (`nq`) prin `0, 1,..., nq-1`; partajăm cele `nV` ore celor `nq` clase, alegând ca toate cele `nS` ore la clasa `k` (`k=0..nq-1`) să fie orele numerotate modulo (`nS`) începând de la `k*nS` - adică orele `k*nS, k*nS+1, k*nS+2,..., k*nS+(nS-1)`. Respectând totuși uzanțele practicii, vom considera în cele ce urmează că numerotarea începe cu 1 (și nu de la 0).
2. ***Graful orelor***: ne imaginăm graful care are ca vârfuri cele `nV` ore, iar ca muchii - perechile de ore care nu se pot desfășura simultan. Aplicând pe acest graf un algoritm de colorare cu `nS` culori, obținem cele `nS` *S*-ore ale săptămânii (vârfurile de o aceeași culoare desemnând un număr de `nV/nS=nq` ore posibile simultan). Observăm că *S*-orele sunt mulțimi independente maximale în acest graf.
3. ***Graful S-orelor:*** considerăm acum graful având ca vârfuri cele `nS` *S*-ore obținute la 2) și ca muchii - acele perechi de *S*-ore care nu s-ar putea desfășura în cadrul unei aceleiași zile (de exemplu, pentru motivul că un același profesor ar trebui să facă într-o aceeași zi, două sau mai multe ore la o aceeași clasă). Colorarea vârfurilor acestui graf cu `nZ` culori, va conduce la obținerea unui orar final (vârfurile de aceeași culoare desemnând un număr de `nS/nZ=nO` *S*-ore posibile simultan, într-o aceeași zi).

Observație

O altă schemă posibilă este următoarea: graful de la 2) se colorează cu `nZ` culori (cele `nO*nq` vârfuri de aceeași culoare desemnează ore posibile simultan într-o aceeași zi); apoi, se trece la colorarea fiecăruia dintre cele `nZ` grafuri rezultate, cu câte `nO` culori (obținând câte un orar pentru fiecare dintre cele `nZ` zile).

O exemplificare

Consolidăm termenii proiectării și maniera de rezolvare prezentată mai sus, abordând o situație concretă: `nq=4` clase, `nZ=2` zile, `nO=3` ore/zi, `nP=5` profesori și următoarea matrice a încadrărilor:

```
2 0 1 1
2 1 0 0
1 2 2 0
0 1 2 2
0 2 1 2
```

Profesorul 1 (linia 1 a matricei) are 2 ore la clasa 1 (coloana 1), 1 oră la clasa 3 și 1 oră la clasa 4; clasa 2 (a doua coloană) are ore respectiv cu profesorii 2 (1 oră), 3 (2 ore), 4 (1 oră) și 5 (2 ore) - în total, 6 ore; clasa 1 are 1 oră fictivă (coloana 1 atribuie în total numai 5 ore din cele `nS=nZ*nO=6` ore).

Conform 1) - cu numerotare de la 1 - partajăm cele `nV=24` ore în `nq=4` clase a câte `nS=6` ore astfel:

```
ora:     1 .. 6   7 .. 12   13 .. 18   19 .. 24
clasa:     1        2          3          4
```

Alocăm pe profesori (în ordinea `1..nP`), orele asociate unei clase; de exemplu, orele `1..6` ale clasei 1 se alocă

astfel: orele 1, 2 - profesorului 1 (în matricea de încadrare, linia 1 are 2 în coloana 1); orele 3, 4 - profesorului 2 (linia 2 are 2 în coloana 1); ora 5 - profesorului 3 (care are 1 oră la clasa 1); ora 6 - profesorului fictiv 0.

Putem formula tabloul complet al asocierilor prin

```
prH[]=(1,1,2,2,3,0,2,3,3,4,5,5,1,3,3,4,4,
       5,1,4,4,5,5,0)
```

unde `prH[i]=` profesorul asociat orei `i`. De exemplu, mulțimea celor 4 ore din încadrarea profesorului 1 este: `{i:prH[i]=1}` = {1,2,13,19};
iar `{i:prH[i]=0}`={6,24} = mulțimea orelor fictive.

Realizând colorarea precizată la 2), putem obține o listă de `nS=6` S-ore - ca în Fig. 3 a); spre exemplu, *S*-ora (4 10 15 23) - a patra din listă - are interpretarea următoare: la clasele 1, 2, 3, 4 pot intra simultan profesorii `prH[4]`=2, `prH[10]`=4, `prH[15]`=3 și `prH[23]`=5. Desigur, *S*-orele obținute vor putea fi exprimate direct prin intermediul tabloului `prH[]` - Fig. 3 b); astfel, putem folosi direct (2 4 3 5), în loc de (4 10 15 23).

(1 7 14 20)	(1 2 3 4)
(2 8 18 21)	(1 3 5 4)
(3 9 13 22)	(2 3 1 5)
(4 10 15 23)	**(2 4 3 5)**
(5 11 16 19)	(3 5 4 1)
(6 12 17 24)	(0 5 4 0)
a) ca ore `1..nV`	b) interpretând prin `prH[]`

Fig. 3 - lista *S*-orelor

Considerând apoi graful celor 6 *S*-ore și colorând vârfurile în `nZ=2` culori - cum am precizat la 3) - putem obține un orar final:

Ziua 1:	1	2	3	4	Ziua 2:	1	3	5	4
	2	3	1	5		2	4	3	5
	3	5	4	1		0	5	4	0

Fig. 4 - un orar final

Alegând - în vederea colorării, la etapa (II) - diverse ordini de parcurgere a orelor `1..nV`, am putea obține mai multe seturi de câte `nS` *S*-ore; unele dintre acestea nu vor putea conduce la un orar final (în etapa 3)), decât slăbind condițiile de adiacență între *S*-ore. Astfel, pe lângă cel din Fig. 3, putem obține și setul de *S*-ore din Fig. 5; presupunând (la fel ca în cazul Fig. 3) că două *S*-ore sunt adiacente (în graful de la 3)) dacă ele conțin un același element într-o aceeași poziție (adică un același profesor, de două ori în ziua respectivă, la o aceeași clasă), atunci rezultă graful din fig. 5.

```
s1=(3 4 5 1)
s2=(2 3 1 4)
s3=(1 3 4 5)
s4=(1 5 3 4)
s5=(2 5 3 0)
s6=(0 2 4 5)
```

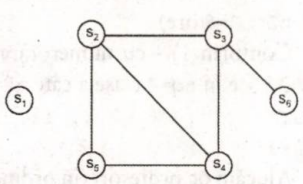


Fig. 5 - Set de *S*-ore și graful asociat

Existența triunghiurilor (s5 - s2 - s4 și respectiv s3 - s2 - s4) face imposibilă colorarea conform 3), cu numai două culori (pentru a obține cele două zile de câte trei *S*-ore). Se pot elimina triunghiurile prin tăierea legăturii s2 - s4 (adică, acceptând ca profesorul 4 să facă 2 ore în aceeași zi la clasa 4); atunci orarul devine posibil: ziua1 = {s2, s4, s6} și ziua2 = {s5, s3, s1}. Mai general, dacă în graful *S*-orelor există un subgraf complet cu `nZ+1` vârfuri, atunci cel puțin un profesor va trebui să aibă într-o anumită zi două ore la o aceeași clasă.

Elaborarea programului

Pentru formulări și clarificări inițiale (pentru ceea ce s-ar numi "versiunea 0") este suficient să se folosească limbajul cel mai obișnuit - Pascal; nu este cazul de a ne gândi (mai întâi) la proiectarea de interfețe și facilități de operare - deocamdată, programatorul ar trebui să se supună nu utilizatorului, ci propriilor interese de profilare și punere la punct a programului.

Este convenabilă o separare în module precum descriem în continuare; în principiu, prima literă a denumirii unit-ului va indica "rangul" acestuia, iar restul identificatorului va evoca funcționalitatea modulului respectiv.

Unit-ul _Const

Este un fișier de configurare minimală, ținând loc de o interfață mai elaborată (destinată transmiterii datelor inițiale).

Listing: _CONST.PAS

```
interface
const
  nq=12;          {          numar de clase }
  nZ=8;           {           numar de zile }
  nO=4;           { numar (maxim) de ore/zi }
  nP=37;          {       numar de profesori }
  fMI='H:\BP70\VB\VB_ORAR\12_37M.TXT';
    {   nume complet pt. fișierul în care }
    {      s-a scris matricea de încadrare }
  fHS='H:\BP70\VB\VB_ORAR\12_37S.TXT';
    { unde se vor depune cele nS "S-ore" }
  {                    ce vor fi obținute }
  fHZ='H:\BP70\VB\VB_ORAR\12_37Z.TXT';
   {        pentru orarul final, în forma }
  {                        "S-ore" pe zile }
  nS=nZ*nO;
  { numar maxim de ore/saptamana la clase }
  nV=nq*nS;     { nr. total de ore (vârfuri }
                {          în graful de la 2)) }
```

Evident, alegerea de dimensiuni constante (modificabile doar prin recompilare) nu ține seama de interesele utilizatorului (dar convine pentru elaborarea unei prime versiuni a programului). Ignorăm aici și necesitatea unui modul de verificare a consistenței datelor primite.

Unit-ul AsetPrCl;

Asigură, conform numerotării de la 2), construcția tablourilor `prH[]` și `clH[]`, pe baza matricei încadrărilor:

`prH[i]:=prof. 1..nP` asociat orei `i=1..nV` (sau, `i:=0` dacă ora `i` este fictivă);

`clH[i]:=clasa 1..nq` corespunzătoare orei `i`.

Fie `setPrCl` procedura care va realiza citirea din fișierul indicat prin `_Const.fMI` și va încărca corespunzător `prH[]` și `clH[]`.

Exercițiu: realizați unit-ul `AsetPrCl`.

Unit-ul BordParc

Pregătește etapa 2), permițând o alegere a ordinii de parcurgere a orelor `1..nV` (în vederea colorării cu `nS` culori, de la 2)).

Listing: BORDPARC.PAS

```
interface
uses _Const, AsetPrCl;
type OrdineParc=(NAT,ORA_CLASE,GRADE,ALTAORD);
var OrdParc:OrdineParc;
    MdH:array[1..nV] of Word;
    Hora:array[1..nV] of Byte;
{ MdH[] conține orele 1..nV într-o ordine }
{       modificată (față de cea naturală) }
{        Hora[j] = "S-ora" în care va fi }
{        repartizată ora a j-a din MdH[] }
procedure pOraClase;
  {       setează MdH[] astfel încât să se }
  { parcurgă ora I-a de la fiecare clasă, }
  { apoi a II-a de la fiecare clasă, etc. }
procedure pGrade;
{ asigură parcurgerea orelor 1..nV ținând }
{ cont de numărul de legături (de gradele }
{  vârfurilor, în graful orelor de la 2)) }
procedure pAltaOrd;
procedure recOrdNat;
{         ordonează Hora[] conform ordinii }
{ naturale 1..nV - încât Hora[j] să indice }
{   "S-ora" orei j și nu pe a orei MdH[j] }
implementation
. . . . . .
Begin
  OrdParc:=NAT;
       { ordinea implicită de parcurgere }
End.
```

Exercițiu: realizați procedurile `pOraClase`, `recOrdNat`; gândiți-vă la o parcurgere a orelor care să nu depindă de ordinea claselor (și a profesorilor).

Folosind una sau alta dintre aceste modalități de parcurgere, se asigură obținerea mai departe a mai multor seturi (foarte diferite) de câte `nS` *S*-ore. Timpul cel mai bun va corespunde (în general) acelei ordini de parcurgere care minimizează numărul de reveniri în arborele de căutare a soluțiilor (pe parcursul cunoscut al aplicării metodei backtracking); de aceea, unit-ul schițat mai sus este punctul critic al programului.

Unit-ul CsetHora

Asigură obținerea celor `nS` *S*-ore, conform 2). Schema generică (*backtracking*) este următoarea:

```
procedure SetHora(nivel:Word);
var COLOR:Word;
begin
  if nivel>nV then STOP:=TRUE
  else
    for COLOR:=1 to nS do
    begin
      OraCurent:=MdH[nivel];
      if OraCurent se poate colora cu COLOR
       { adică este posibilă simultan cu  }
       { orele colorate anterior cu COLOR }
       then
       begin
         Hora[nivel] := COLOR;
         SetHora[nivel+1);
         if STOP then Exit
       end
    end
end;
```

unde `STOP` este o variabilă a unit-ului, inițializată cu `FALSE` și care servește pentru rezumarea procesului de căutare la (numai) prima soluție.

Exercițiu: realizați procedura `SetHora`.

Exercițiu: presupunând că s-a ajuns la `nivel=nV+1` (deci s-a obținut prima soluție) - de câte ori se va executa `end` (respectiv, cele trei instrucțiuni `end` din finalul textului de mai sus)? Gândiți-vă la modalități de a forța revenirea *directă* (în punctul din care s-a apelat `setHora(1)`.

Unit-ul DwrHore;

Prin procedura `WriteHore`, înscrie cele `nS` *S*-ore obținute (notând profesorii `0..nP` asociați orelor conținute în *S*-ore) în fișierul indicat de `_Const.fHS`.

Exercițiu: realizați `WriteHore`.

Depunând *S*-orele obținute într-un fișier, se rezervă utilizatorului posibilitatea de a modifica (dacă se pricepe) unele dintre *S*-ore, înainte de a continua cu etapa 3).

Unit EgetHore

Furnizează tabloul

`HoS:array[1..nS,1..nq] of Byte;`

în care (prin procedura `GetHore`) citește *S*-orele existente în fișierul `_Const.fHS`.

Exercițiu: realizați `GetHore`.

Exercițiu: gândiți-vă la o procedură de ordonare corespunzătoare a *S*-orelor din `Hos[]` (încât colorarea care urmează la 3) să se obțină într-un timp cât mai bun).

Unit FsetZiHO

Furnizează tabloul `ZiH:array[1..nS] of Byte;` care va fi completat de procedura de colorare:

`SetZiHo(nivel:Word)`

astfel încât `ZiH[j]` = culoarea (ziua) atribuită *S*-orei a `j`-a din `HoS[]`.

Exercițiu: realizați `SetZiHo()` (analog cu `CsetHora.SetHora()`).

Unit-ul GwrZile

Prin procedura `WriteZile`, înscrie orarul final obținut (sub forma "Ziua", *S*-orele acelei zile), în fișierul `_Const.fHZ` (folosind `ZiH[]` și `Hos[]`).

Exercițiu: realizați `WriteZile`.

Unit-urile suplimentare HlistaPr și IminFer

Sunt destinate formatării orarului obținut și respectiv, minimizării numărului de ferestre.

Exercițiu: realizați funcția `ListaOre(p:Byte)` care să returneze (ca **`string`**) lista orelor profesorului `p`;

Exemplu

```
1: |-DA-B-| |DAB---| ...
```

ar însemna că profesorul 1 are în prima zi ore la clasele D (a II-a din zi), A (a III-a), B (a V-a) și are fereastră ora a IV-a; în a doua zi are primele trei ore, la clasele notate D, A, B; etc.

Exercițiu: realizați o funcție de extragere și listare a orarului unei clase precizate.

Exercițiu: realizați o funcție care să dea numărul de ferestre dintr-o zi (din cadrul orarului obținut anterior). Realizați o procedură care să ordoneze *S*-orele dintr-o zi astfel încât numărul de ferestre să fie minim.

Asamblarea modulelor

Programul vizat de modulele descrise mai sus are forma următoare:

Listing MAIN.PAS

```
uses _Const,
{       dimensiuni, matricea de incadrare }
     AsetPrCl,
{         numerotarea orelor, conform 1)   }
     BordParc,
{     ordinea repartizarii orelor in S-ore }
     CsetHora,
{ obtine S-orele, conform (II) si BordParc }
     DwrHora,
     { pune la dispozitie S-orele obtinute }
     EgetHore, { incarca S-orele (eventual }
          {          modificate manual) }
     FsetZiHo,
          { repartizeaza S-orele pe zile, }
          {                    conform 3) }
     GwrZile; {     scrie orarul final - nZ
}
          { secvente de câte nO "S-ore }
Begin
  SetPrCl;
  SetHora(1);
  WriteHore;
  GetHore;
  SetZiHO(1);
  WriteZile
End.
```

Se va putea considera un alt program, pentru minimizarea numărului de ferestre în orarul obținut și pentru formatarea orarului:

Listing: MAIN2.PAS

```
uses _Const, HlistaPr, IminFer;
var s:Byte;
Begin
  GetOrar;  { citeste orarul din _Const.fHZ
}
  for s:=1 to nZ do
    MinimizNrFer(s);{ reordonează S-orele }
     { din ziua s, incat numarul total de }
    {                ferestre să fie minim }
    PreiaDenumiri;          { nume profesori,
}
    {          denumiri clase, nume materii }
    WrOrarPr;          { listeaza orele pe
}
    {        profesori, apeland ListaOre() }
    WrCondica    { listeaza orarul sub forma
}
  {                    obisnuita in condica }
...
End.
```

Bibliografie

1. V. Bazon - *O clasă de pătrate latine și aplicații*, G.M.m. nr. 1-2/1983
2. V. Bazon - *Factorizarea grafului K2n și aplicații*, G.M.m. nr. 1/1987
3. V. Bazon - *Limbaje și calculator*, Ed. Petrion 1998
4. N. Christofides - *Graph theory - an algorithmic approach*, 1975
5. L. Livovschi, H. Georgescu - *Sinteza și analiza algoritmilor*, 1986

GĂLĂCIUC '99

Doru Popescu Anastasiu

În luna iulie s-a desfășurat la Gălăciuc, județul Vrancea Concursul anual de informatică pentru Gimnaziu, desfășurarea concursului fiind posibilă datorită efortului depus de dl. Emil Onea, inspector la ISJ Vrancea. Vă prezentăm în continuare problemele propuse la acest concurs.

Muzica pe CD (clasa a V-a)

Se consideră `n` ($1 \le n \le 100$) melodii ale căror durate de audiere (în minute și secunde) sunt cunoscute. Există un CD pe care se pot înregistra melodii cu o durată totală egală cu `ttm` ($0 \le ttm \le 59$) minute și `tts` ($0 \le tts \le 59$) secunde. Duratele melodiilor sunt date sub forma a doi vectori `tm=(tm(1), tm(2), ..., tm(n))` și `ts=(ts(1), ts(2), ..., ts(n))`, `tm` pentru minute și respectiv `ts` pentru secunde. Componentele lui `tm` și `ts` sunt numere naturale cuprinse între `0` și `59`. Astfel melodia `1` are `tm(1)` minute și `ts(1)` secunde, melodia `2` are `tm(2)` minute și `ts(2)` secunde și așa mai departe.

Se cere:

a) Să se afișeze durata tuturor melodiilor în minute și secunde.

b) Să se înregistreze pe CD cât mai multe melodii (nu se pot înregistra melodii incomplete) astfel încât între oricare două melodii să existe un interval de 5 secunde pauză.

Se vor afișa numerele asociate melodiilor, precum și timpul rămas neînregistrat de la ultima melodie până la sfârșitul CD-ului.

Observații

1) Este posibil să existe mai multe soluții. În acest caz se va afișa una singură.
2) Nu are importanță ordinea în care se înregistrează melodiile pe CD.
3) Vă rugăm să respectați formatul de afișare a datelor de ieșire din exemplul de mai jos.

Exemplu

Intrare

```
n=8
ttm=21
tts=4
tm=(4,2,3,2,6,4,6,4)
ts=(10,5,0,5,7,10,7,10)
```

Ieșire

```
a) 31 minute si 54 secunde
b) melodiile pe CD sunt:
1 2 3 4 6 8
timpul ramas este
1 minute si 24 de secunde
```

Soluție

Pentru punctul a) calculăm suma componentelor vectorului `tm` în `sm` și suma componentelor vectorului `ts` în `ss`, după care transformăm în minute secundele date de `ts`, adunându-le la `sm` și scăzându-le din `ss`.

Pentru punctul b) ordonăm melodiile crescător după durata lor, urmând să le afișăm pe primele care au durata totală mai mică sau egală cu cea a CD-ului și maximă cu această proprietate.

Listing: MUZICA.PAS

```
Program muzica_pe_CD;
var tm,ts,o:array[1..100]of Integer;
    n,ttm,tts,i,sw,aux:Integer;
    s,tcd,sm,ss:Longint;

Begin
  Write('n=');Readln(n);
  Write('ttm=');Readln(ttm);
  Write('tts=');Readln(tts);
  for i:=1 to n do
  begin
    Write('tm(',i,')=');
    Readln(tm[i]);
    Write('ts(',i,')=');
    Readln(ts[i]);
    o[i]:=i
  end;
  sm:=0;
  ss:=0;
```

```
  for i:=1 to n do
  begin
    sm:=sm+tm[i];
    ss:=ss+ts[i]
  end;
  sm:=sm+ss div 60;
  ss:=ss mod 60;
  Writeln('a) ',sm,' minute si ',ss,' secunde');
  sw:=1;
  while sw=1 do
  begin
    sw:=0;
    for i:=1 to n-1 do
    if tm[i]>tm[i+1]
    then
    begin
      aux:=tm[i]; tm[i]:=tm[i+1];
      tm[i+1]:=aux; aux:=ts[i];
      ts[i]:=ts[i+1]; ts[i+1]:=aux;
      aux:=o[i]; o[i]:=o[i+1];
      o[i+1]:=aux; sw:=1
    end
    else
      if tm[i]=tm[i+1] then
        if ts[i]>ts[i+1] then
        begin
          aux:=tm[i]; tm[i]:=tm[i+1];
          tm[i+1]:=aux; aux:=ts[i];
          ts[i]:=ts[i+1]; ts[i+1]:=aux;
          aux:=o[i]; o[i]:=o[i+1];
          o[i+1]:=aux; sw:=1
        end
  end;
  Writeln('b) melodiile pe CD sunt: ');
  s:=0; i:=1; tcd:=tts+ttm*60;
  while (s<=tcd)and(i<=n)do
  begin
    Write(o[i],' ');
    s:=s+tm[o[i]]*60+ts[o[i]];
    i:=i+1
  end;
  Writeln('timpul ramas: ');
  Writeln((s-tcd)div 60,' minute si ',
                (s-tcd)mod 60,' de secunde')
End.
```

Codificare (clasa a VI-a)

Se dă de la tastatură o cifră k diferită de 0 și 1 și un text x (cu cel mult 255 caractere) format din litere mici ale alfabetului englez. Textul x se codifică literă cu literă, fiecare literă fiind înlocuită cu un număr natural egal cu k^{poz}, unde poz este poziția literei în alfabet. Astfel, 'a' se codifică prin k^1, 'b' se codifică prin k^2, și așa mai departe. Între numerele codificării se lasă câte un spațiu.

Se cere să se afișeze cea mai lungă secvență (litere succesive) de litere egale din x, precum și codificarea textului.

Observații

1. La punctul a), dacă există mai multe soluții se va afișa una singură.
2. Datele de intrare se consideră corecte.
3. Se va respecta formatul de afișare a datelor de ieșire din exemplu.

Exemplu

Pentru k=2 și x='gaaala' se va afișa:

```
a) aaa
b) 128 2 2 2 4096 2
```

Soluție

Pentru punctul a) determinăm pentru fiecare literă din text secvența cea mai lungă care are toate literele egale cu ea (și începe cu ea). După care o afișăm pe cea mai lungă (prima).

Pentru punctul b) folosim funcția poz_lit care returnează poziția unei litere în alfabet, iar pentru calculul puterii k^{poz} folosim vectori, deoarece aceste puteri pot depăși tipurile numerice predefinite.

Listing: CODIFIC.PAS

```
Program codificare;
var x,z,zz:string;
    transport,p,lung_max:Integer;
    aux,i,j,n,k,poz,dim_a:Integer;
    a:array[1..1000]of Integer;

function poz_lit(lit:Char):Integer;
begin
  poz_lit:=ord(lit)-96
end;

Begin
  Write('k=');Readln(k);
  Write('x=');Readln(x);
  { punctul a) }
  n:=Length(x);
  poz:=1;
  lung_max:=1;
  for i:=1 to n do
  begin
    j:=1;
    while (i+j<=n)and(x[i+j]=x[i]) do
      j:=j+1;
    if lung_max<j
    then
    begin
      poz:=i; lung_max:=j
    end
  end;
  Write('a) ');
  for i:=poz to poz+lung_max-1 do
    Write(x[i]);
  Writeln;
  { punctul b) }
```

```
  for i:=1 to n do
  begin
    poz:=poz_lit(x[i]);
    dim_a:=1; a[1]:=1;
    for j:=1 to poz do
    begin
      transport:=0;
      for p:=1 to dim_a do
      begin
        aux:=a[p]*k+transport;
        a[p]:=aux mod 10;
        transport:=aux div 10
      end;
      if transport<>0 then
      begin
        dim_a:=dim_a+1; a[dim_a]:=transport
      end
    end;
    z:='';
    for j:=dim_a downto 1 do
    begin
      Str(a[j],zz); z:=z+zz
    end;
    Write(z,' ')
  end
End.
```

Numere asemenea (clasa a VII-a)

Se dă o mulțime X={x(1), x(2), ..., x(n)} de n (1≤n≤100) numere naturale mai mici sau egale cu 10^5 și un număr natural y mai mic sau egal cu 10^5.

Două numere naturale scrise în baza 10 a(1)a(2)... a(p) și b(1)b(2)...b(q) se numesc *asemenea* dacă p=q și există k∈{-9, -8, . . ., 8, 9} astfel încât a(i)=b(i)+k, pentru orice i∈{1,2,...,n}.

a) Să se afișeze acele numere din X care sunt asemenea cu y.

b) Să se afișeze o submulțime Z a lui X cu număr maxim de elemente care să aibă proprietatea că orice două elemente din Z sunt asemenea.

Exemplu

Intrare

```
n=9
X={15, 213, 38, 324, 49, 546, 200, 657, 30}
y=27
```

Ieșire

```
a) 38 29
b) 213 324 546 657
```

Soluție

În rezolvarea acestei probleme vom folosi două funcții, nr_cifre și asemenea, prima returnând numărul de cifre ale unui număr (dat ca parametru), iar a doua verifică dacă două numere (date ca parametri) sunt asemenea.

Listing ASEMENEA.PAS

```
Program numere_asemenea;
var x:array[1..100]of Longint;
    i,j,n,po,lu,nu:Integer;
    y:Longint;

function nr_cifre(a:Longint):Integer;
var nr:Integer;
begin
  nr:=0;
  while a<>0 do
  begin
    nr:=nr+1; a:=a div 10
  end;
  nr_cifre:=nr
end;

function asemenea(a,b:Longint):Boolean;
var p,q,kk,dim,i,j:Integer; sw,bec:Boolean;
    x1,y1:array[1..10]of 0..9;
begin
  p:=nr_cifre(a); q:=nr_cifre(b);
  sw:=false;
  if p=q
  then
  begin
    dim:=0;
    while a<>0 do
    begin
      dim:=dim+1;
      x1[dim]:=a mod 10;
      y1[dim]:=b mod 10;
      a:=a div 10; b:=b div 10
    end;
    for kk:=-9 to 9 do
    begin
      bec:=true;
      for j:=1 to dim do
        if x1[j]<>kk+y1[j]
        then bec:=false;
      if bec then sw:= true
    end
  end;
  asemenea:=sw
end;

Begin
  Write('n=');Readln(n);
  for i:=1 to n do
  begin
    Write('x[',i,']=');
    Readln(x[i])
  end;
  Write('y=');
  Readln(y);
  Write('a) ');
```

```
  for i:=1 to n do
    if asemenea(x[i],y)
    then Write(x[i],' '); Writeln;
  lu:=0;
  for i:=1 to n do
  begin
    nu:=0;
    for j:=i+1 to n do
      if asemenea(x[i],x[j])
      then nu:=nu+1;
    if nu>lu
    then
    begin po:=i; lu:=nu end
  end;
  Write('b) ',x[po],' ');
  for i:=po+1 to n do
    if asemenea(x[po],x[i])
    then Write(x[i],' ')
End.
```

Piața

Se dă o piață dreptunghiulară pavată cu m linii și n coloane ($1 \le m, n \le 20$) de dale pătratice; unele dale sunt verzi.

a) În această piață se pot efectua doar pași de anumite lungimi introduse de la tastatură. Se poate păși doar pe dale verzi, iar deplasările trebuie să fie paralele cu marginile pieței. Dala de coordonate (1,1) se află în colțul din stânga sus a pieței. Dându-se coordonatele unei dale de plecare și ale unei dale de sosire, se cere să se găsească o secvență de pași prin care să se ajungă de la plecare la sosire. Fiecare deplasare va fi codificată prin lungimea pasului și direcția în care este efectuat acest pas (*Nord*, *Sud*, *Est*, *Vest*). Dalele de plecare și sosire sunt verzi.

Observație

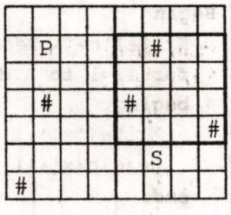
Numărul de lungimi de pași si numărul de dale verzi se introduc de la tastatură.

b) În piață trebuie așezată o tarabă pătrată de latură p dată, astfel încât să fie acoperite cât mai multe dale verzi. Laturile tarabei vor fi paralele cu marginile pieței. Taraba nu are voie să iasă în afara pieței. La ieșire se va preciza numărul de dale acoperite.

Exemplu

Intrare

```
m=7
n=8
p=4
nr. lungimi: 5
lung. pasilor : 1 2 3 4 7
nr. dale verzi: 8
coordonate dale verzi:
2 5  2 6  4 2  4 5  5 8  7 1  2 2  6 6
punctul de start: 2 2
punctul de sosire: 6 6
```



Ieșire

```
a) 2Sud 3Est 2Nord 1Est 4Sud
b) 4
```

Soluție

Pentru punctul a) folosim metoda *backtracking* și determinăm o singură soluție. Codificarea celor patru tipuri de deplasări *Nord*, *Sud*, *Est*, *Vest* este precizată prin vectorii de constante, x și y. Secvența de pași este reținută în tabloul t, fiecare componentă t_i având 4 informații: numărul mutării, tipul deplasării dat de vectorii x și y, precum și lungimea acesteia.

Pentru punctul b) se glisează taraba pe tabloul corespunzător pieței și se reține poziția care acoperă cele mai multe dale verzi.

Listing PIATA.PAS

```
Program piata;
const x:array[1..4] of -1..1=(-1,0,1,0);
      y:array[1..4] of -1..1=(0,1,0,-1);
type punct=record
             p:byte;
             x,y,z:Integer
           end;
var a:array[1..20,1..20] of 0..1;
    i,j,m,n,nr_pasi,nr_verzi,i1,j1:Integer;
    is,js,ip,jp,p,max,nr:Integer;
    z:array[1..20]of Integer;
    t:array[1..20,1..20] of punct;
    bec:Boolean;

procedure afisare(i,j,nr:Integer);
begin
  if nr>1 then
  begin
    afisare(i-t[i,j].z*t[i,j].x,j-
                    t[i,j].z*t[i,j].y,nr-1);
    if (t[i,j].x=-1)and(t[i,j].y=0)
    then Write(t[i,j].z,'Nord ');
    if (t[i,j].x=0)and(t[i,j].y=1)
    then Write(t[i,j].z,'Est ');
    if (t[i,j].x=1)and(t[i,j].y=0)
    then Write(t[i,j].z,'Sud ');
    if (t[i,j].x=0)and(t[i,j].y=-1)
    then Write(t[i,j].z,'Vest ')
  end
end;  { afisare }

procedure back(iii,jjj,pas:Integer);
var k,h,ii,jj:Integer;
begin
  if not bec
  then
    for h:=1 to nr_pasi do
      for k:=1 to 4 do
```

```
      begin
        ii:=iii+x[k]*z[h];
        jj:=jjj+y[k]*z[h];
        if(ii in [1..m])and(jj in [1..n]
                          and not bec
       then if(a[ii,jj]=1)and (t[ii,jj].p=0)
            then
            begin
              t[ii,jj].p:=pas;
              t[ii,jj].x:=x[k];
              t[ii,jj].y:=y[k];
              t[ii,jj].z:=z[h];
              if(ii=is)and(jj=js)
              then
              begin
                Write('a) ');
                afisare(is,js,t[is,js].p);
                bec:=true
              end
              else back(ii,jj,pas+1);
              t[ii,jj].p:=0
            end
      end
end;  { back }

Begin
  Write('m=');Readln(m);
  Write('n=');Readln(n);
  Write('p=');Readln(p);
  Write('nr. lungimi pasi: ');
  Readln(nr_pasi);
  Write('lungimile pasilor: ');
  for i:=1 to nr_pasi do
  Read(z[i]);
  Write('nr. dale verzi: ');
  Readln(nr_verzi);
  for i:=1 to m do
    for j:=1 to n do
    begin
      a[i,j]:=0;
      t[i,j].p:=0
    end;
  Writeln('coordonate dale verzi: ');
  for i:=1 to nr_verzi do
    begin
      Read(i1,j1);
      a[i1,j1]:=1
    end;
  Write('punctul de start: ');
  Readln(ip,jp);
  Write('punctul de sosire: ');
  Readln(is,js); t[ip,jp].p:=1;
  bec:=false;
  back(ip,jp,2);
  Writeln; max:=0;
  for i:=1 to m-p+1 do
    for j:=1 to n-p+1 do
    begin
      nr:=0;
      for i1:=0 to p-1 do
        for j1:=0 to p-1 do
          if a[i+i1,j+j1]=1 then inc(nr);
          if max<nr then max:=nr
    end;
  Writeln('b) ',max);
End.
```

Au câştigat...

Clasa a V-a

Nicola Laurenţiu, Dolj
Tăutu Bogdan, Bucureşti
Ioniţă Lucian, Suceava

Clasa a VI-a

Stancu-Mara Sorin, Bucureşti
Grigoraş Paul, Suceava
Ioniţă Andrei, Olt

Clasa a VII-a

Stoenescu Teodor, Bucureşti
Preoteasa Alexandru, Dolj
Popa Răzvan, Bacău

Clasa a VIII-a

Pufu Silviu, Argeş
Berinde Radu, Bucureşti
Prodan Sebastian, Alba

Felicitări !!!

TOAMNA se numără...

...ALGORITMII

Sorin Groza

Venirea toamnei coincide și cu începerea unui nou an școlar, deci un nou început, o nouă perioadă de muncă pentru îmbogățirea cunoștințelor în ale informaticii. Pentru un succes cât mai mare este nevoie de o bază cât mai solidă. În acest scop continuăm prezentarea unor probleme împreună cu algoritmii corespunzători.

Greierele și furnicile

Pe toată durata verii, greierele a cântat și a jucat și s-a împrumutat de la vecinele furnici. Iată că acum, în prag de iarnă acestea s-au gândit să-i ceară fiecare datoria. Cum greierele nu este învățat cu calculele s-a gândit că are nevoie de un program pentru a afla ceea ce-l interesează, adică:

- cea mai mare sumă cu care este dator și la câte furnici trebuie să dea această sumă;
- care este cel mai mare număr de furnici care trebuie să primească o aceeași sumă de bani de la greiere și care este această sumă.

Exemplu:

Intrare

```
n=8
25 99 12 25 99 25 88 12
```

Ieșire

```
suma maxima datoare 99 la 2 furnici
3 furnici - suma 25
```

Pentru a afla cea mai mare sumă cu care este dator greierele, vom determina valoarea maximă din șirul datoriilor (`max`) și vom vedea de câte ori este prezentă această valoare în șir (contorul `k`). A doua cerință este echivalentă cu aflarea valorii care apare de cele mai multe ori în șir. Vom lua fiecare element a_i și-l vom compara cu toate elementele a_j ($j=i, n$) în caz de egalitate incrementând un contor `k`. Dacă `k` este mai mare decât maximul determinat până în acel moment, valoarea lui va fi stocată în `max` iar valoarea lui a_i va fi stocată în `val`. În final `max` va indica numărul maxim de valori egale, iar `val` va indica valoarea respectivă.

Listing GREIERE.PAS

```
const lim=100;
var a:array[1..lim] of Integer;
    max,n,k,i,j,val:Integer;
Begin
  Write('n='); Readln(n);
  for i:=1 to n do Read(a[i]);
  for i:=1 to n do
    if a[i]>max then max:=a[i];
  for i:=1 to n do if a[i]=max then k:=k+1;
  Writeln('suma maxima datoare ',max,
                        ' la ',k,' furnici');
    { cautarea valorii celei mai frecvente }
  max:=0;
  for i:=1 to n-1 do
  begin
    k:=0;
    for j:=i to n do
      if a[i]=a[j] then k:=k+1;
    if k>max then begin max:=k; val:=a[i] end
  end;                          { sfarsit cautare }
  Writeln(max,' furnici - suma ',val)
End.
```

Aranjarea cuvintelor

La gura sobei doi copii joacă următorul joc: fiecare îi propune celuilalt un număr de cuvinte și încă un cuvânt suplimentar. Fiecare trebuie să aranjeze cuvintele unul sub altul, astfel încât citite pe o coloană `k` să se obțină cuvântul suplimentar. Ei au observat că pentru multe cuvinte și de lungime mare, găsesc foarte greu soluția jocului. Scrieți-le un program care afișează cum trebuie așezate cuvintele și pe ce coloană trebuie făcută citirea pentru a se obține cuvântul suplimentar.

Exemplu:

Intrare

5 - *nr. cuvinte*

tren cult mac acar teama - *cuvintele mulțimii*

careu - *cuvântul suplimentar*

Ieșire

```
Coloana de citire 2
a|c|ar
m|a|c
t|r|en
t|e|ama
c|u|lt
```

Pentru a vedea dacă pe o coloană k se poate obține cuvântul suplimentar procedăm astfel: vom forma un nou cuvânt prin lipirea literelor de pe poziția k din fiecare cuvânt. Mai rămâne să vedem dacă mulțimea literelor acestui nou cuvânt este identică cu mulțimea literelor cuvântului suplimentar. Pentru aceasta se ordonează alfabetic cele două cuvinte și se verifică dacă sunt identice. Dacă da, se iau pe rând literele cuvântului suplimentar și pentru fiecare literă se găsește cuvântul care are pe poziția k litera respectivă. Este clar că numărul maxim al coloanei este egal cu lungimea celui mai mic dintre cuvinte (min). Deci $1 \le k \le$ min. Dacă nu se găsește potrivirea pentru nici o coloană se va afișa un mesaj corespunzător.

Listing CUVINTE.PAS

```
const max=255;
var a:array[1..50] of string;
    b,c,d,e:string; aux:Char;
    i,j,x,y,n,min:Integer; sol:Boolean;

Begin
  Write('Numarul de cuvinte: '); Readln(n);
   Writeln('Introduceti cuvintele multimii:
');
  min:=max;
  for i:=1 to n do
  begin
    Readln(a[i]);
    if Length(a[i])<min
    then min:=Length(a[i])
  end;
  Write('Introduceti cuvantul: ');Readln(b);
  c:=b;      { ordonare cuvant suplimentar }
  for j:=1 to Length(c) do
  begin
    for i:=1 to Length(c)-1 do
    begin
      if c[i]>c[i+1]
      then
      begin
        aux:=c[i];
        c[i]:=c[i+1];
        c[i+1]:=aux
      end
    end
  end;                    { sfarsit ordonare }
  for i:=1 to min do
  begin
    e:='';
    for j:=1 to n do e:=e+a[j][i];
      { ordonarea cuvant de pe o coloana }
    for x:=1 to Length(e) do
    begin
      for y:=1 to Length(e)-1 do
      begin
        if e[y]>e[y+1]
        then
        begin
          aux:=e[y];
          e[y]:=e[y+1];
          e[y+1]:=aux
        end
      end
    end;                  { sfarsit ordonare }
    if e=c
    then
    begin
      sol:=true;
      Writeln('Coloana de citire ',i);
      for x:=1 to Length(b) do
      begin
        for y:=1 to n do
          if b[x]=a[y][i] then Writeln(a[y])
      end
    end
  end;
  if not sol then Writeln('Fara solutie!')
End.
```

Cursul dolarului

În fiecare zi putem afla cu toții raportul dintre dolarul american și leul românesc. Față de ziua precedentă, dolarul poate valora mai mult sau mai puțin. Cei care urmăresc cursul dolarului sunt interesați de tot felul de statistici. Având cotația dolarului pentru o perioadă de n zile se cere să se scrie un program care găsește cea mai lungă perioadă de zile în care creșterea dolarului este constantă. Programul va afișa pe ecran perioada de cotații căutată și creșterea în această perioadă.

Exemplu:

Intrare

```
n=12
5 2 15 7 10 13 16 81 4 8 12 1
```

Ieșire

```
Perioada de cotatii cautata este: 7 10 13 16
Cresterea zilnica este 3
```

Pentru cei care nu au aflat până acum, secvența cerută reprezintă o progresie aritmetică. Algoritmul folosește faptul că diferența dintre oricare două elemente vecine ale unei progresii aritmetice este aceeași. Din această cauză, vom construi un alt șir b după regula $b_i=a_{i+1}-a_i$ pornind de la șirul inițial. Aflarea secvenței cerute din șirul a este echivalentă cu aflarea celei mai lungi secvențe de elemente egale din șirul b. Acest lucru se face prin compararea elementelor vecine și creșterea unui contor cât timp acestea sunt egale. Când găsim o valoare mai mare decât cea găsită până la acel moment vom păstra poziția de început și de sfârșit a secvenței.

Listing DOLAR.PAS

```
var a,b:array[1..50] of Integer;
    max,i,k,j,n,p1,p2,salt:Integer;

Begin
  Write('n='); Readln(n);
  for i:=1 to n do Read(a[i]);
           { crearea sirului cu diferente }
  for i:=1 to n-1 do b[i]:=a[i+1]-a[i];
  b[n]:=-1;                  { sfarsit creare }
  k:=0;
  for i:=1 to n do
  begin
    if b[i]=b[i+1] then k:=k+1
    else
    begin
      if k>max
      then
      begin
        max:=k; p1:=i-max;
        p2:=i+1; salt:=b[i]
      end;
      k:=0
    end
  end;
  if p1=0 then Writeln('Nu exista solutie!')
  else
  begin
    Write('Perioada de cotatii cautata este:');
    for i:=p1 to p2 do Write(a[i],' ');
    Writeln;
    Writeln('Cresterea zilnica este ',salt)
  end
End.
```

În încheiere vă propunem spre rezolvare câteva probleme, cerându-vă algoritmi cât mai eficienți.

Grupuri

La o croazieră participă n turiști și nu există decât un vapor de capacitate dată c. În vederea constituirii grupurilor ce vor călători pe rând cu vaporul, ghidul oferă bilete pe care sunt notate litere din alfabet. Un grup este format din acele persoane care au tras bilete cu litere identice. Să se spună dacă toate grupurile formate în urma tragerii biletelor au în componență un număr mai mic de persoane decât capacitatea vaporului. În caz negativ se va afișa mesajul "IMPOSIBIL".

Grupurile vor pleca în ordinea descrescătoare a numărului de componenți.

Să se afișeze grupurile în ordinea în care vor pleca în croazieră.

Exemplu:

Intrare

```
n=10
A B C A B L A C B H
c=6
```

Ieșire

```
(1,4,7)
(2,5,9)
(3,8)
(6)
(10)
```

Bâlbâitul

Un candidat cam bâlbâit dictează un articol, purtătorului său de cuvânt. Bâlbâitul are tendința de a repeta unele cuvinte. Purtătorul de cuvânt, în vederea transmiterii articolului către ziar, va trebui să elimine toate repetările succesive ale unui cuvânt. Scrieți un program care afișează articolul corect.

Exemplu:

Intrare

```
eu am am sa fac o o o sosea ca la la carte.
```

Ieșire

```
eu am sa fac o sosea ca la carte.
```

Balul

La marele bal organizat de *Ziua Dăruinței* sosesc b băieți și f fete. Se cunosc înălțimile tuturor celor b băieți și celor f fete. O pereche de dansatori trebuie formată dintr-o fată și un băiat de aceeași înălțime. Se dorește să se formeze un număr maxim de perechi pentru dans. Să se afișeze pe ecran câte perechi vor dansa, precum și componența acestora.

Exemplu:

Intrare

```
b=4
140 125 140 130
f=5
120 140 130 125 110
```

Ieșire

```
3 perechi
1 cu 2; 2 cu 4; 4 cu 3
```

I. CĂLĂTORIA premianților

În perioada 11-25 Noiembrie 1999 s-a desfășurat excursia organizată de Ministerul Tineretului și Sportului pentru medaliații români la diferite olimpiade internaționale în ultimii doi ani. A participat și redactorul-șef adjunct al Gazetei de Informatică, medaliat cu argint la Olimpiada Internațională de Informatică din 1998. Dacă doriți să aflați mai multe despre această excursie citiți "impresiile de călătorie" care urmează.

La această excursie au participat 26 de olimpici internaționali dintre care zece informaticieni, cinci matematicieni, patru biologi, trei fizicieni, trei chimiști și un filozof. Excursia a costat aproximativ 60.000$, deci a însemnat un efort financiar apreciabil din partea organizatorilor. Pe lângă olimpici au participat și persoane reprezentând Ministerul Tineretului și Sportului, Ministerul Educației Naționale, firma CISCO Systems etc.

Orașele vizitate în timpul excursiei au fost Paris, Los Angeles, Las Vegas și San Francisco. Pentru informaticieni excursia a fost interesantă deoarece am vizitat câteva firme din domeniul IT&C, am participat la expoziția COMDEX și ne-am întâlnit cu mai multe persoane importante din acest domeniu.

Se poate spune că acțiunea *Călătoria premianților 1999* a fost o reușită din punctul de vedere al Ministerului Tineretului și Sportului care a reușit să obțină aproape tot ce și-a propus în această excursie și anume informarea olimpicilor asupra posibilităților de studiu în Statele Unite ale Americii, schimbarea mentalității unor români plecați în SUA cu privire la situația actuală a României, scoaterea în evidență a performanțelor obținute de tinerii români în diferite situații.

Cele mai importante puncte din programul excursiei au fost vizitele la patru universități americane (UCLA, Caltech, Northridge și Stanford), la diferite firme de software și hardware (Apple Computers, CISCO Systems, GigaPixel etc.), precum și la expoziția COMDEX.

În cele ce urmează vă voi împărtăși impresiile acestei călătorii, în viziunea unui informatician.

Participanți:

INFORMATICĂ:
- Bogdan Batog, București
- Adrian Cârcu, Bistrița
- Bogdan Dumitru, Ploiești
- Tudor Leu, Focșani
- Daniel Oprean, Cugir
- Mihai Pătrașcu, Craiova
- Angel Proorocu, București
- Mihai Scorțaru, Cluj
- Mihai Stroe, București
- Radu Ștefan, Brașov

MATEMATICĂ:
- Marius Beceanu, Iași
- Dorian Dobrin, Pitești
- Adrian Ioana, Târgu Jiu
- Mihai Manea, Galați
- Valentin Tonița, Bacău

BIOLOGIE:
- Cristian Baboilă, Craiova
- Liliana Diaconu, București
- Teodora Duna, Tulcea
- Anita Stoica, Bacău

FIZICĂ:
- Mihai Amariei, Suceava
- Bogdan Marinoiu, Ploiești
- Constantin Tănasă, Iași

CHIMIE:
- Cristian Grădinaru, Suceava
- George Iancu, Buzău
- Dănuț Maceș, Tulcea

FILOZOFIE:
- George Tudorie, Urziceni

Miercuri, 10 noiembrie '99

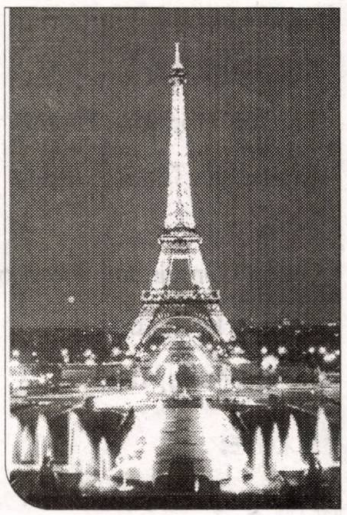
Totul a început în staţia CFR din Cluj-Napoca în jurul orei 21:35 când s-a făcut auzit anunţul: "Trenul rapid 634 pleacă peste cinci minute în direcţia Câmpia Turzii, Războieni, [...] Bucureşti Nord. Vă rugăm poftiţi în vagoane!". Acesta este momentul în care a început "aventura" numită *Călătoria premianţilor 1999*. Trec aproximativ 15 minute, dar călătorii nu reacţionează deoarece sunt obişnuiţi cu întârzierile CFR. Totuşi "gluma se îngroaşă" în momentul în care se face auzit un nou anunţ: "Trenul rapid 634 este reţinut în staţia Cluj până la noi ordine!". În acest moment pasagerii se precipită spre funcţionari şi primesc diferite explicaţii cu privire la motivul întârzierii; până la urmă aflăm că angajaţii Căilor Ferate Române au hotărât să intre în grevă şi că nu vor relua lucrul decât în cazul în care ministrul transporturilor îşi prezintă demisia. Totuşi suntem rugaţi să aşteptăm "până la noi ordine". Noile ordine ajung la Cluj în jurul orei 23:00 când se face auzit anunţul "paralizant": "Trenul rapid 634 este anulat!". Cam aşa a început excursia olimpicilor pentru mai mulţi dintre participanţii care nu se aflau încă în Bucureşti pe data de 10 Noiembrie. Până la urmă, la fel ca şi ceilalţi, am reuşit şi eu să ajung în Capitală cu sprijinul firmei Electric 21 Service.

Joi, 11 noiembrie '99

Am ajuns la sediul Ministerului Tineretului şi Sportului în jurul orei 8:00; au început să sosească participanţii la excursie care şi-au rezolvat problema deplasării la Bucureşti în cele mai originale moduri. Am fost informaţi că trebuie să participăm la o conferinţă de presă organizată în cinstea noastră la care vor participa dl. Crin Antonescu, Ministrul Tineretului şi Sportului şi Excelenţa Sa dl. James Rosapepp, ambasadorul Statelor Unite ale Americii la Bucureşti.

Conferinţa de presă a început cu "discursurile" celor doi invitaţi care ne-au felicitat pentru rezultatele obţinute, ne-au rugat să fim demni ambasadori ai României în Statele Unite şi ne-au urat călătorie plăcută. În continuare ar fi urmat întrebările ziariştilor prezenţi, dar cum aceştia s-au arătat deosebit de "interesaţi" de olimpicii români, singura întrebare a venit din partea reprezentantei MEDIAFAX care a dorit detalii suplimentare cu privire la excursie. Am fost felicitaţi încă o dată şi am plecat spre Aeroportul Internaţional Bucureşti-Otopeni pentru a pleca spre Paris. Ne-am îmbarcat pe un avion Airbus al companiei Air France, iar zborul spre Paris a durat aproximativ trei ore.

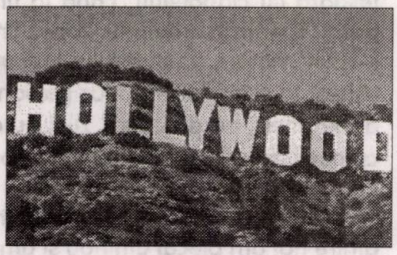
La plecare am fost surprinşi de sunetul unui clopoţel care indica direcţia în care trebuie să mergem. Până la urmă ne-am dat seama că o doamnă profesoară care ne însoţea în excursie a uitat cu cine are de-a face (probabil că în opinia dânsei noi aveam între 7 şi 10 ani) şi a considerat că s-ar putea să ne pierdem prin aeroport.



La Paris am fost cazaţi în apropierea aeroportului Charles de Gaulle la o distanţă de peste 30 de kilometri de oraş. Ni s-a spus că dacă dorim neapărat să vedem Parisul acest lucru este posibil cu condiţia să ne întoarcem la o anumită oră. Cei care am decis să mergem la Paris am ajuns în centrul oraşului unde am fost informaţi că vom vizita Arcul de Triumf, Muzeul Louvre şi ne vom plimba pe Champs d'Elysée şi pe malul Senei. Totuşi ni s-a oferit opţiunea de a vizita singuri Parisul cu condiţia să ne întoarcem într-un anumit loc. Unii au profitat de ocazie şi au reuşit să viziteze şi alte monumente interesante cum ar fi Turnul Eiffel, biserica Notre Damme de Paris, palatul Elysée etc. În final am aflat că cei care au respectat programul oficial nu au reuşit să ajungă nici măcar la Arcul de Triumf, unde avea loc un spectacol interesant cu ocazia Zilei Armistiţiului (încheierea primului război mondial – 11 Noiembrie 1918).

Vineri, 12 noiembrie '99

În dimineața acestei zile sunetul clopoțelului ne-a condus spre aeroport de unde am plecat spre Los Angeles tot cu un avion al companiei Air France. Zborul a durat aproximativ unsprezece ore și, datorită diferenței de fus orar, aceasta a fost cea mai lungă zi din viața majorității participanților, ea având 33 de ore.



După îndeplinirea formalităților vamale ne-am deplasat la hotelul unde eram cazați în orașul Hollywood, celebra suburbie a Los Angeles-ului. Din fața hotelului se putea vedea emblema de pe dealul din apropierea Hollywood-ului, cunoscută de aproape toată lumea. Am aflat și eu că Hollywood, Beverly Hills etc. nu sunt cartiere ale Los Angeles-ului, ci orașe independente care sunt considerate suburbii.

Seara, unii au făcut o scurtă plimbare prin apropiere și s-au declarat foarte dezamăgiți de modul în care începuse aventura pe pământ american.

Sâmbătă, 13 noiembrie '99

Aceasta poate fi considerată ziua în care a început cu adevărat excursia. Dimineața am vizitat sediul UCLA (University of California Los Angeles) unde trebuia să ne întâlnim cu diferite personalități locale. Până la urmă am fost primiți ca oricare grup de turiști, ne-am plimbat prin campus și am aflat diferite detalii despre modul în care putem studia aici. Am fost impresionați de biblioteca acestei universități unde putea fi găsită orice carte publicată în Statele Unite, precum și de librăria UCLA unde erau prezente ultimele cărți apărute în cele mai diverse domenii. Spre sfârșitul vizitei ni s-a spus că în această universitate predau trei laureați ai premiului Nobel.

A urmat vizita la Universal Studios, locul în care sunt realizate majoritatea filmelor hollywood-iene. Aici am găsit un adevărat oraș (numit Universal City) care era, de fapt, un mic parc de distracții.

După un tur al acestui oraș, în care am fost victimele unei inundații, am văzut cum un pescar este înghițit de un rechin, am căzut într-un vulcan, am rămas blocați în metrou, ne-am întâlnit cu King Kong și, mai presus de toate, am văzut diferite trucuri folosite la filmări. A continuat plimbarea prin Universal City; am vizitat locuri ca Jurassic Park, Back to the Future, Terminator II 3D, ET etc. Ne-am întâlnit și cu ciocănitoarea Woody care era prezentă peste tot în Universal City.

Seara am servit cina la Transilvania Grill, singurul restaurant românesc din Los Angeles. A participat și consulul României la Los Angeles precum și diferiți membri ai comunității românești din L.A. Am fost felicitați și aici pentru rezultatele obținute și am primit un cadou simbolic din partea consulului. Au urmat discuții cu diferiți români plecați în SUA, majoritatea dintre ei sfătuindu-ne să pierdem avionul de întoarcere.

După cină, patru dintre noi am fost invitați să vizităm orașul; am reușit să vedem Pacificul, ne-am plimbat pe Rodeo Street, strada pe care își fac cumpărăturile celebritățile din Beverly Hills și am aflat mai multe detalii (neplăcute) despre modul de viață american. (În momentul în care am trecut printr-o zonă rău famată, însoțitorii noștri ne-au asigurat că nu avem motive de îngrijorare și pentru a ne liniști ne-au arătat armamentul din dotare.)

Duminică, 14 noiembrie '99

De această dată clopoţelul ne-a condus spre cel mai celebru parc de distracţii din lume – Disneyland. Ajunşi aici ne-am dat seama că nu prea avem ce face aici deoarece toate distracţiile erau destinate copiilor sub 10 ani (probabil că ştia ceva doamna profesoară cu clopoţelul). Până la urmă ne-am plimbat şi noi prin parc, am vizitat diferite zone "interesante" cum ar fi ţara lui Mickey Mouse, ţinuturile lui Indiana Jones, diferite dealuri şi munţi cu nume ciudate şi seara am asistat la o defilare a unor personaje de desene animate de care nu am mai auzit. Spre surprinderea mea, pe parcursul zilei nu am văzut nicăieri în parc vreun Mickey Mouse care să se joace cu copiii (nu erau prezenţi nici Donald şi nici vreun alt personaj celebru).
După această zi plictisitoare, unul dintre românii care a făcut parte din grupul care ne-a arătat în ziua precedentă L.A.-ul, s-a oferit să ne arate acum lucruri cu adevărat interesante în oraş. Trei dintre noi am plecat din nou şi am vizitat alte două locuri pe malul Pacificului şi am mers în diferite locuri în care se întâlnesc tinerii din L.A. Am vizitat şi sediul unei firme la care era angajat însoţitorul nostru, precum şi locuinţa unei familii de români stabiliţi în SUA. Aici am aflat că în Statele Unite convorbirile telefonice locale sunt gratuite, motiv pentru care ne-am fi putut petrece o mare parte a timpului navigând pe Web; totuşi am decis că există în Los Angeles lucruri mult mai interesante decât acces cu adevărat gratuit la Internet.

Luni, 15 noiembrie '99

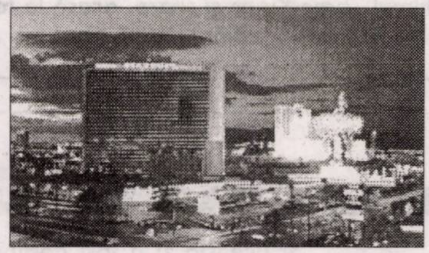
În dimineaţa zilei am fost anunţaţi că programul de week-end se terminase şi urmează un program mai interesant. Acesta a început cu o vizită la redacţia ziarului Los Angeles Times. Din motive "necunoscute", începând cu această zi, sunetul clopoţelului nu s-a mai auzit.

Vizitând sediul LA Times, am aflat că acesta este cel mai important cotidian local din Statele Unite ale Americii, că se află pe locul doi în SUA ca tiraj (1.100.000 exemplare) după New York Times, că numărul din ziua precedentă (duminică) avea 872 de pagini şi costa 1.50$ şi că un număr obişnuit are aproximativ 100 de pagini şi costă 0.25$. Spre bucuria mea, am observat că tehnoredactorii de la LA Times folosesc aceleaşi instrumente software ca şi tehnoredactorii de la GInfo; totuşi echipamentele hardware erau net superioare...
Acesta a fost programul nostru interesant din ziua de luni; după vizita la LA Times am plecat spre Las Vegas, călătoria cu autocarul pus la dispoziţie de organizatori durând aproximativ şase ore. Pe drum a avut loc o adevărată "bătaie cu gheaţă" organizată de o parte a informaticienilor, rezultatul luptei fiind dezastruos pentru toţi cei implicaţi.
La Las Vegas am vizitat diferite cazinouri cum ar fi Paris, Caesar, Hilton etc. şi am urmărit diferite spectacole interesante. Cel mai mult am fost impresionaţi de dansul fântânilor arteziene din centrul oraşului care ofereau un spectacol extraordinar. Într-o altă parte a Las Vegas-ului era simulată erupţia unui vulcan tot prin intermediul unor fântâni arteziene şi a unui joc de lumini. Obişnuiţi deja cu ciudăţeniile americanilor nu ne-a mai surprins faptul că dacă doream să urcăm în replica turnului Eiffel de la cazinoul Paris, trebuia să plătim cam de zece ori mai mult decât dacă am fi urcat în turnul din Paris. Un eveniment de care îşi vor aminti mulţi dintre noi s-a petrecut într-un cazino, unde un domn a pierdut în faţa noastră 25.000$ la o aruncare cu zarul.
Din păcate nici unul dintre olimpici nu a avut dreptul să-şi încerce norocul, deoarece există o limită de vârstă (21 de ani) pentru a participa la jocurile din cazinouri şi autorităţile americane ţineau la respectarea strictă a acestei prevederi. Totuşi au existat numite încălcări ale legislaţiei americane, comise de unii dintre membrii grupului nostru în această zi, dar care au adus beneficii numai statului american.

Marți, 16 noiembrie '99

Aceasta a fost ziua în care a început cu adevărat programul "academic", din păcate numai pentru informaticieni. Dimineața am asistat la prezentarea susținută de președintele CISCO Systems, dl. John Chambers, care a descris planurile companiei sale pentru următorii ani, a prezentat câteva dintre noile produse și ne-a informat de existența Academiei CISCO. Pentru prezentarea activităților academiei, dl. Chambers, a fost "ajutat" de o membră a grupului nostru, d-ra Maria Zamfir, elevă în clasa a XII-a la Liceul "Dimitrie Cantemir" din București.

A urmat ceea ce așteptau informaticienii și anume vizita la COMDEX, una dintre cele mai importante expoziții din domeniul IT&C. Dintre cele trei clădiri ale expoziției, în această zi noi am reușit să vizităm doar una. Totuși am văzut câteva dintre ultimele noutăți în domeniu cum ar fi sistemul de operare Windows CE, sisteme echipate cu procesoare Intel Pentium III la 733 MHz sau AMD Athlon la 700 MHz, diferite LCD-uri, plăci video (cipul GeForce 256 de la nVidia a fost starul în acest domeniu), notebook-uri echipate cu procesoare Intel Pentium III, inscriptoare de CD-uri care atingeau viteze de până la 12X, sau CD drive-uri cu viteze de citire de până la 60X, diferite dispozitive de stocare etc. Am participat la diferite concursuri și am obținut premii foarte "interesante" dintre care cele mai apreciate au fost pixurile cu ajutorul cărora făceam baloane de săpun.



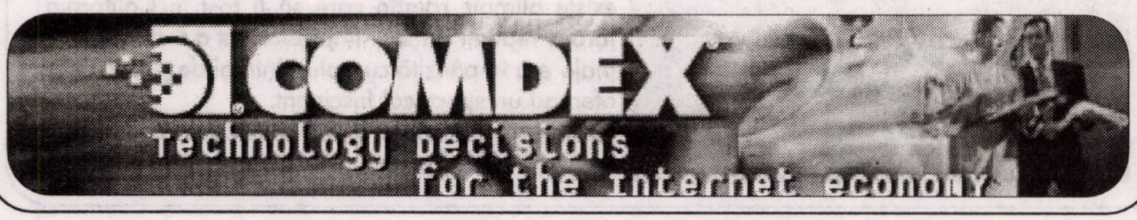
Miercuri, 17 noiembrie '99

Organizatorii ne-au înțeles dorința de cunoaștere și au hotărât să petrecem o parte din această zi tot la COMDEX. Dimineața am avut o întâlnire cu câteva personalități din cadrul firmei CISCO Systems, precum și cu unul dintre organizatorii COMDEX. Acesta din urmă ne-a prezentat în câteva cuvinte istoria de 20 de ani a expoziției și a purtat cu noi discuții pe diferite teme cum ar fi microprocesoare, plăci video, pachete software etc.

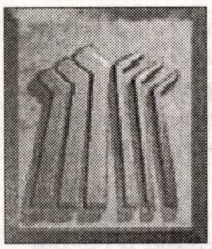
În următoarele câteva ore am putut vizita din nou expoziția, așa că am plecat într-una dintre cele două clădiri nevizitate anterior. Aici am văzut standurile unor firme importante cum ar fi Microsoft, Xerox, Canon, Hewlett Packard, Ericsson, Sony, Philips, CISCO Systems etc. În standul Xerox ne-am întâlnit cu Nadia Comăneci și Bart Conner care au fost angajați de respectiva companie pentru a oferi autografe vizitatorilor.

Nu am mai avut timp să vizităm și a treia clădire care era destinată în totalitate sistemului de operare LINUX, peste 30 de distribuitori ai acestui sistem de operare prezentându-și produsele. Am fost surprins de faptul că nu existau standuri ale celor mai mari producători de microprocesoare, atât Intel cât și Advanced Micro Devices (AMD) refuzând invitația de a participa la expoziție. Așa cum am mai spus, produsele acestor firme au apărut în standurile altor companii care ofereau sisteme complete sau plăci de bază, dar, oficial, nu au existat reprezentanți Intel sau AMD la COMDEX.

Spre seară am pornit din nou spre Los Angeles unde am ajuns după șase ore. De această dată Orașul Îngerilor nu mai era atât de impresionant deoarece ne aminteam cu toții de spectacolul din Orașul Luminilor.



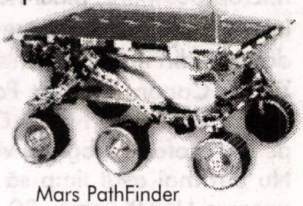
Joi, 18 noiembrie '99



În această zi a continuat programul academic cu o vizită la CSUN (California State University of Northridge), o universitate mediocră dar la care preda o profesoară de origine română. Aici am fost primiți de diferite persoane care ne-au prezentat laboratoarele universității, modul de predare, precum și modalități de înscriere. Cei de la CSUN se lăudau cu un robot produs de IBM care reușea să efectueze diferite operații cum ar fi deplasarea unui braț și ridicarea unor obiecte; angajații universității au fost foarte suprinși când li s-a spus că la Universitatea Tehnică din Cluj-Napoca există un robot proiectat și realizat de un grup de studenți care are performanțe mult superioare.
Ne-am întâlnit și cu o familie de români care ne-au invitat pe șase dintre noi să petrecem noaptea în locuința lor. Ne-am mai plimbat puțin prin L.A., am fost și la un cinematograf unde am vizionat filmul *Three Kings* și seara, acasă la famila de români, ne-am jucat cu diferitele viețuitoare care locuiau acolo: câine, pisică, hamster, papagal și ... șarpe.

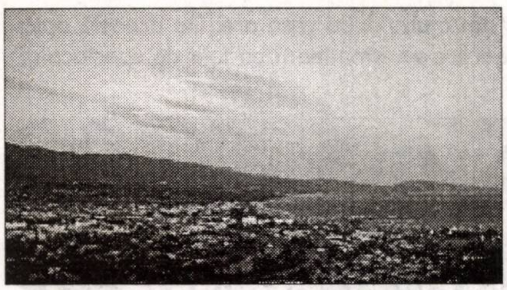
Vineri, 19 noiembrie '99

Și această zi a fost destinată vizitării unei universități și anume Caltech (California Institute of Technology), o replică la mult mai celebrul MIT (Massachusetts Institute of Technology) din Boston. Totuși aici am putut vedea câteva lucruri interesante și ne-am întâlnit cu persoana răspunzătoare cu admiterea studenților la această universitate, dl. John Latting.
Am vizitat și laboratorul JPL (Jet Propulsion Laboratory), finanțat de NASA, majoritatea persoanelor importante de aici fiind și profesori la Caltech. Am aflat detalii despre diferitele proiecte JPL, unul în curs de finalizare fiind misiunea Mars Polar Lander (o sondă va *amartiza* la unul din polii planetei roșii, iar imaginile transmise de această sondă vor putea fi vizionate în direct pe Internet. Probabil că în momentul în care citiți aceste rânduri misiunea a luat deja sfârșit. (Amartizarea are loc pe data de 3 Decembrie 1999).
Programul prevedea și o vizită în laboratorul de ceramică de la Caltech, vizită pe care cei mai norocoși dintre noi am reușit să o evităm.
Seara, aproximativ zece dintre cei care nu reușiseră să vadă încă Pacificul au fost invitați de un grup de studenți români de la Caltech să viziteze orașul Los Angeles.



Mars PathFinder

Sâmbătă, 20 Noiembrie '99

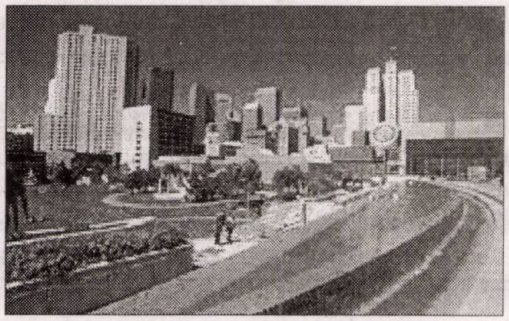


Această zi a fost destinată călătoriei spre San Francisco care a durat aproximativ 10 ore. Pe parcurs ne-am oprit în Santa Barbara pentru ca să nu existe olimpic român care să fi fost în California fără a vedea Pacificul. Impresionant a fost faptul că plaja era împânzită cu stoluri întregi de păsări care ofereau un spectacol fascinant.
Spre seară am ajuns în San Francisco, dar nu am reușit să vedem prea multe în oraș.

Duminică, 21 noiembrie '99

Dimineața ne-am întâlnit cu câțiva dintre membrii comunității române din San Francisco. Locul de întâlnire a fost stabilit în fața unei biserici construite de românii stabiliți aici, dar care nu semăna de loc cu o biserică ortodoxă. Preotul vorbea perfect limba română, dar avea o manieră foarte interesantă de a ține slujba, pe care eu nu am mai întâlnit-o în nici o biserică ortodoxă din România. Biserica era construită potrivit legilor statului California, deci avea și două ieșiri de urgență care să poată fi observate de toată lumea. După slujbă preotul ne-a informat că în timpul săptămânii s-au strâns 4.200$ și ne-a rugat să contribuim la rambursarea împrumutului contractat pentru construirea bisericii.

După acest spectacol care i-a surprins pe mulți am fost invitați "oficial" în biserică și am fost prezentați membrilor comunității române din San Francisco. Aceștia ne-au felicitat și ei, ne-au urat succes în continuare, ne-au sfătuit să rămânem în SUA, iar apoi câțiva dintre ei s-au oferit să ne însoțească într-o vizită a orașului. Am vizitat celebrele poduri din San Francisco, am trecut pe lângă închisoarea federală de pe insula Alcatraz și am aflat lucruri noi despre oraș.



Seara am fost invitați la cină în locuința unui român stabilit în San Francisco, locuință care putea fi destul de ușor confundată cu un palat.

Luni, 22 noiembrie '99

A fost din nou o zi plină pentru informaticieni, deoarece am vizitat celebrul Silicon Valley. Aici am aflat mai multe despre modul în care poți să-ți întemeiezi o companie IT; nu e nevoie decât de o idee interesantă și anumiți investitori (numiți *angel investors*) o vor finanța.

Am vizitat apoi o astfel de companie condusă de un evreu născut în România, companie numită GigaPixel. Aceasta era cea de-a treia companie înființată de acest domn în Silicon Valley, precedentele două fiind cumpărate de alte companii în momentul în care erau pe punctul de a deveni publice (cotate la bursă). Compania GigaPixel a proiectat o placă video numită GP-1; ne-au fost prezentate comparativ plăcile GP-1 și una bazată pe cipul GeForce 256 de la nVidia și, la prima vedere, se părea că GP-1 este mult superioară. Totuși olimpicii români nu s-au lăsat prea ușor păcăliți și au observat că jocul pentru care era făcută demonstrația (Quake III) nu rula la aceeași rezoluție pe ambele calculatoare, că sistemul de operare folosit nu era același, frecvențele procesoarelor erau diferite, toate aceste diferențe favorizând GP-1. Totuși, placa video se baza pe o idee revoluționară care se pare că va aduce câteva milioane de dolari firmei GigaPixel și se pare că mulți au fost păcăliți la prezentarea comparativă între GP-1 și GeForce 256.

A urmat o vizită la compania Apple Computers, unde am fost însoțiți de un angajat Apple de origine română. Acesta ne-a prezentat ultimele calculatoare MacIntosh din familia G4 și ne-a arătat diferite obiective din campusul Apple. Dânsul afirma că un procesor produs de Apple este de două ori mai rapid decât un procesor Intel la aceeași frecvență. Personal am avut ocazia să testez un procesor Apple la 400 MHz și mi s-a părut mai lent decât un Pentium III la 450 MHz. Am fost informați că procesoarele produse de Apple ating frecvențe de până la 450 MHz și că sistemele MacIntosh echipate cu astfel de procesoare ar fi echivalente cu PC-uri la 900 MHz care nu există încă. Dacă respectivul domn ar fi fost mai bine informat ar fi aflat că începând cu 15 Noiembrie firma Kriotech oferă procesoare Athlon cu viteze de 1000 MHz.

Seara am putut vizita locuința unui român angajat la Apple Computers, și aceasta putând fi confundată cu un palat.

Marți, 23 noiembrie '99

Aceasta a fost ultima noastră zi în Statele Unite ale Americii; am vizitat o ultimă universitate americană și anume Stanford. Aici am fost primiți de diferiți profesori care predau la această universitate printre care și o profesoară de origine română. Dânsa ne-a prezentat diferite documente istorice originale care se află în posesia acestei universități, multe dintre ele referitoare la momente importante din istoria României. Până la urmă și-a dat seama că nimeni nu este interesat de așa ceva și ne-a permis să ne continuăm vizita prin campus. Ne-am întâlnit și cu studenți români și am aflat și detalii referitoare la modalitatea de înscriere la Stanford.

Ar fi trebuit să vizităm și sediul firmei Hewlett Packard dar împrejurările au fost de așa natură încât acest lucru nu a mai fost posibil. Așa că am pornit spre aeroportul din San Francisco, de unde am plecat spre Paris tot cu un avion al companiei Air France, călătoria durând aproximativ zece ore. Probabil că aceasta a fost cea mai scurtă zi din viață multora dintre noi, datorită diferenței de fus orar, ea având doar 15 ore.

Miercuri, 24 noiembrie '99

Am ajuns la Paris și ni s-a oferit din nou posibilitatea vizitării orașului în aceleași condiții ca și prima dată. Deși unii dintre noi ar fi dorit să vadă orașul până la urmă majoritatea au renunțat. Din păcate, această seară petrecută de noi la Paris poate fi comparată cu o seară obișnuită într-un hotel oarecare dintr-un oraș oarecare, într-o țară oarecare.

Joi, 25 noiembrie '99

Zborul între aeroporturile Charles de Gaulle și București-Otopeni a durat aproape patru ore, tot compania Air France fiind cea la care am apelat. În timpul zborului s-au schimbat numere de telefon, adrese de e-mail, cărți de vizită în speranța că vom păstra legătura între noi. Ajunși la București ne-am despărțit și ne-am împrăștiat prin toate colțurile țării. De data aceasta trenul rapid 633 a plecat în direcția [...] Cluj-Napoca și a ajuns la destinație cu o întârziere de numai 20 de minute.

PREMIUL LUNII

Microsoft România oferă în această lună abonaților 3 pachete MSDN Universal Subscription

MSDNTM Subscriptions vă ajută să beneficiați de avantajele celor mai recente instrumente și tehnologii Microsoft. Cu sistem de abonare flexibil, organizat pe mai multe nivele și avantajul de fi actualizat la timp pe tot parcursul anului, MSDN Subscription este esențial pentru orice dezvoltator.

MSDN Universal Subscription

MSDN Universal Subscription face mai ușoară dezvoltarea unor soluții particulare oferind toate informațiile, tehnologia și instrumentele de dezvoltare de care aveți nevoie. Este pur și simplu modul cel mai eficient, și ieftin de a dezvolta și testa soluții.

Prin flexibilitatea unor nivele de abonament adaptate fiecăruia și a avantajului unor actualizări la timp de-a lungul întregului an, un abonament MSDN este esențial pentru orice dezvoltator.

Abonamentul de tip Universal include toate facilitățile unuia de tip Professional plus toate produsele apărute de-a lungul anului, incluzând: Microsoft Visual Studio 6.0 Enterprise Edition, BackOffice Test Platform, instrumente de dezvoltare suplimentare, versiuni internaționale, produse și instrumente în curs de apariție, actualizări și produse nou-lansate.

Aceasta a fost *Călătoria premianților 1999*, cel puțin atât cât se poate scrie în paginile unei reviste. Totuși olimpicii au organizat și un concurs (fără premii), al cărui clasament final îl puteți vedea în chenarul alăturat. Dacă bănuiți în ce a constat concursul vă rugăm să trimiteți pe adresa redacției ipotezele dumneavoastră. Datorită faptului că acest concurs nu a fost unul cu premii, probabil că nici dumneavoastră nu veți fi premiați, dar... cine știe?

Probabil că mulți dintre dumneavoastră se întreabă de ce a apărut un astfel de articol în GInfo. Evident că așa ceva nu ar avea ce căuta într-o astfel de revistă, dar redactorii GInfo au și ei anumite privilegii. Sperând că nu v-ați supărat prea mult pe mine, că numărul cititorilor nu s-a redus drastic în urma publicării acestui articol, că unii dintre dumneavoastră înțeleg utilitatea lui, vă mulțumesc pentru că ați citit despre evenimentele petrecute în Statele Unite, vă urez să ajungeți și voi acolo (dacă nu ați fost deja) și să vă dați seama că, totuși, este mai bine să trăiești în România.

Clasament

Dumitru Bogdan	☺☺☺☺☺☺
Adrian Cârcu	☺☺☺☺
Tudor Leu	☺☺☺

Impresiile altora...

Angel Stelian Proorocu

argint, BOI'99
argint, CEOI'98
bronz, BOI'97

"Cele două săptămâni au trecut. Au rămas amintirile și, ceea ce este mult mai important, au rămas prieteniile care s-au legat. Cred că mai mult decât peisajele unice, mai mult decât o imagine din San Francisco, Los Angeles sau Las Vegas, mult mai mult contează oamenii pe care i-am întâlnit, ideile și învățămintele pe care le-am putut decanta după această experiență unică, interesantă atât din punct de vedere turistic, cât și academic."

Pentru o excursie ca aceasta merită să înveți chiar și... geometria analitică!

Tudor Ioan Leu

argint, BOI '99
argint, IOI '98
bronz, CEOI '98
bronz, BOI '97

"Cea mai interesantă zi a fost cea petrecută în Silicon Valley. A fost interesantă pentru că acolo am putut vedea foarte bine ce înseamnă «lumea de mâine». Cineva mi-a spus: «Dacă ești cu adevărat bun, acesta este unul din locurile în care poți să ajungi», și probabil avea dreptate.

Dumitru Bogdan

aur, IOI '99
argint, CEOI '99

"Pentru mine, excursia în Statele Unite ale Americii a însemnat în primul rând un prilej de cunoaștere a unei lumi noi, a unui mod de viață cu totul diferit de cele cunoscute până acum. Am avut ocazia să discut cu membri ai comunității românești din California, cu studenți români de la prestigioase universități americane, simțind astfel un sentiment românesc în inima Americii."

Anonymous

"Fără Ministerul Tineretului și Sportului, nu aș fi văzut America!"

II. SOFTWARE

A apărut SuSE Linux 6.3

Pe data de 1 decembrie 1999 a fost lansată o nouă versiune SuSE Linux, care dispune de un nou program de instalare grafică, facilitând utilizarea acestui sistem de operare. Cu un DVD sau 6 CD-ROM-uri, SuSE Linux 6.3 furnizează un sistem de operare Linux actualizat și sigur, având incluse peste 1500 de aplicații, garantând, de asemenea, o instalare foarte ușoară celor neinitiați în Linux. Printre cele 1500 de aplicații pentru Linux oferite se numără și programe cunoscute cum ar fi Apache, Samba, Sendmail, GIMP și RealPlayer 5.0. Utilizatorii vor beneficia de 230 de programe noi precum și de 180 de pachete reactualizate. Pentru cei interesați, noul SuSE Linux 6.3 poate fi comandat direct la adresa http://www.suse.de/en/produkte/order/index.html sau la distribuitorii și dealer-ii autorizați ai companiei SuSE.

Linux în China

Guvernul chinez este atât de încântat de spiritul care domină comunitatea "open source", încât va face din Linux sistemul de operare oficial al Republicii Populare Chineze, prin semnarea unui contract cu compania americană GraphOn. Acest contract va permite unei varietăți de aplicații să fie folosite, la costuri reduse, în universitățile din China, în sistemul militar și chiar în interiorul guvernului. Conform declarațiilor anumitor oficiali ai companiei GraphOn, există o puternică asemănare între China comunistă și comunitatea "open source".

Violența din jocuri

Chiar în momentul în care industria jocurilor electronice se pregătește de venirea celui mai important sezon de cumpărături, criticii violenței care caracterizează multe dintre aceste jocuri își fac din nou auzite opiniile. Doi senatori ai Statelor Unite, Joseph Liebman și Herb Kohl, împreună cu un grup care militează împotriva violenței din jocurile electronice, Institutul Național pentru Media și Familie, au făcut public un raport anual care tratează aceasta problemă. Ținta predilectă a acestor critici a fost "Kingpin: Life of Crime" produs de Interplay Entertainment Corporation, în care jucătorii trebuie să se "transforme" în criminali de stradă.

Solaris 8

Firma Sun Microsystems a anunțat în această săptămână o versiune preliminară a sistemului de operare Solaris 8 și a declarat că va lansa oficial noul produs în luna februarie a anului 2000, în apropierea datei de lansare a sistemului de operare Microsoft Windows 2000. Oficialii Sun speră să atragă utilizatorii de Windows NT și Linux către Solaris 8, versiunea preliminară fiind disponibilă pe Internet la un preț de 19,99 de dolari, plus taxele poștale. Costul acoperă atât materialele cât și documentația necesară.

Cluster Linux bazat pe Itanium

Prima demonstrație a unui cluster Linux bazat pe un procesor Itanium (viitorul procesor Intel pe 64 de biți) marchează un punct de turnură în industria calculatoarelor. SGI a făcut o demonstrație a primului cluster Linux bazat pe procesorul Intel IA-64 Itanium, folosindu-se de ocazia oferită de târgul Supercomputing '99. Demonstrația a inclus o serie de premiere, alături de primul cluster bazat pe un procesor Itanium apărând și primele aplicații tehnice rulate cu succes pe o platformă IA-64/Linux. Demonstrația s-a bazat în totalitate pe componente "open source" inclusiv Cactus (un 'toolkit' pentru simulări), MM5 (o simulare meteo multidimensională), Amira (un sistem avansat de vizualizare 3D și modelare a volumului), un compilator C, un compilator Fortran etc.

Microsoft Office 2000, pe Internet

Confirmând tendința generală de a furniza aplicațiile și serviciile legate de acestea prin Internet, oficialii Microsoft au declarat că pachetul Office 2000 va fi disponibil într-o versiune găzduită pe Internet. Microsoft Office Online, cum se va numi această versiune, este destinată mai ales afacerilor mici și mijlocii care nu au resurse să administreze singure softul. Portalul Microsoft pentru afaceri mici și mijlocii, numit "bCentral.com", va fi cel care va oferi serviciul, dar la fel vor face și alți furnizori de servicii pentru aplicații (ASP). Microsoft Office Online va putea rula pe clienți care folosesc sistemele de operare Windows 9x, Windows CE, Windows NT Terminal Server Software și viitorul Windows 2000 Terminal Service Software. Printre aplicațiile oferite utilizatorilor se numără: Microsoft Word, Microsoft Excel, Microsoft Powerpoint, Microsoft FrontPage, Microsoft Outlook, Microsoft Access și altele.

Windows Millennium beta 2

La sfârșitul lunii noiembrie Microsoft a lansat versiunea Beta 2 a sistemului de operare Windows Millennium. Atenția maximă a viitorului sistem de operare va fi "rețeaua casnică", declară oficialii companiei. Versiunea Beta 2 include o lărgire a tehnologiilor Internet Connection Sharing, care facilitează formarea rețelelor casnice, dar oferă și un suport mai larg pentru echipamentele hard ale rețelelor. De asemenea apare și prima implementare a suportului "Universal Plug and Play". Versiunea Millennium a sistemului de operare Windows va fi lansată probabil în cea de-a doua jumătate a anului viitor și va fi probabil ultimul sistem de operare bazat pe un kernel Windows 9x.

Aplicații pentru W2K

S-ar părea că Microsoft a impus standarde aproape inaccesibile pentru acordarea dreptului de utilizare a logo-ului Windows 2000 în cadrul aplicațiilor. Până în acest moment o singură aplicație, OmniPage 10 de la Caere Corp., a trecut testele oficiale efectuate în Los Angeles de VeriTest Inc. În plus, VeriTest nu a făcut publice testele aplicațiilor pentru server, o problemă și mai mare din cauza gamei foarte mari și complexe de aplicații pe care Windows 2000 Server trebuie să le suporte. Această întârziere înseamnă că foarte puține aplicații vor putea fi oficial folosite cu Windows 2000 în momentul lansării acestuia.

Noul Win NT 4.0 SP6

Compania Microsoft a lansat o versiune modificată a pachetului Service Pack 6 pentru sistemul de operare Windows NT 4.0. Acesta este destinat rezolvării problemei apărute în implementarea Winsock din versiunea originală SP6. În practică eroarea de programare afecta utilizatorii care încercau să folosească anumite aplicații server-based; aceștia se confruntau cu erori de genul *Could not connect to server* sau *Acces denied*. Noul SP6 înlocuiește fișierul AFD.SYS și rezolvă alte câteva probleme minore.

Upgrade-uri la Windows 2000

În ultimele săptămâni, Microsoft a oferit clienților și partenerilor săi informații despre viitorul sistem de operare ce va fi lansat pe 17 februarie 2000, mai ales referitoare la costurile acestuia. Schema de prețuri a Microsoft oferă diferite puncte de plecare pentru fiecare produs, restul depinde de cantitate, de platforma de la care se pornește și alte amănunte. Astfel, Windows 2000 Professional va costa 219 dolari dacă reprezintă un upgrade de la Windows '95 sau '98, 149 de dolari dacă este un upgrade de la Windows NT 4.0 Workstation și 319 dolari altfel. Pentru piața serverelor, compania din Seattle a declarat că prețurile practicate vor fi la jumătatea celor prezente pe piață.

III. HARDWARE

AMD Athlon la 750 MHz

Firma Advanced Micro devices a preluat din nou conducerea în "războiul procesoarelor", lansând pe data de 29 noiembrie 1999 un nou procesor Athlon având frecvența de 750 MHz. Acest procesor este primul produs AMD bazat pe tehnologia de 0.18 microni, tehnologie care permite micșorarea dimensiunilor procesorului, a consumului de energie și creșterea vitezei. Prețul de fabrică pentru un astfel de procesor este de 799$ pentru comenzi mai mari de 1000 de bucăți.

AMD K6-2 la 533 MHz

AMD a preluat conducerea și în lupta pentru piața procesoarelor cu preț scăzut, lansând pe data de 29 noiembrie 1999 un nou procesor K6-2 având frecvența de 533 MHz. Între timp, firma Compaq a hotărât să folosească acest procesor în viitoarele sale produse, fapt care ar putea duce la impunerea AMD în acest domeniu. Procesorul este construit folosind tehnologia de 0.25 microni, iar prețul de fabrică este de 167$ pentru comenzi de peste 1000 de bucăți.

Intel vs. AMD

Cursa spre bariera de 1 GHz continuă, principalii competitori fiind Intel și AMD, ambele firme anunțând că preconizează livrarea de procesoare la 1 GHz în cursul anului următor. AMD afirmă că procesorul Athlon va atinge viteza de 1 GHz în a doua jumătate a anului 2000, până atunci urmând să apară versiuni la 800 MHz (primul trimestru al anului 2000) și 900 Mhz (al doilea trimestru al anului 2000). Se pare că procesoarele Pentium III bazate pe actuala tehnologie de 0.18 microni vor putea atinge cu ușurință viteza de 900 MHz și pot fi și ele "împinse" până la 1000 MHz.
Va fi interesant de urmărit duelul din cursul anului următor când Intel și AMD vor lupta în continuare pentru supremație în "războiul procesoarelor". AMD promite lansări succesive ale procesoarelor Athlon la viteze de 800, 850 și 900 MHz înainte de a atinge bariera de 1 GHz. De asemenea, Intel va lansa înainte de a ajunge la 1 GHz, procesoare având frecvențe de 750, 800, 866 și 933 MHz.
Din această "bătălie" va ieși învingător consumatorul deoarece va trebui ca cele două companii să reducă prețurile produselor pentru a rezista în fața concurenței.

E2K, nou intrat în "războiul procesoarelor"

Banca de investiții Robert Flemings a declarat de curând că speră să poată aduna până la mijlocul anului viitor fondurile necesare unor ingineri electroniști ruși, care spun că au dezvoltat un microprocesor care depășește performanțele oricărui produs executat în vestul Europei sau SUA. Inginerii au declarat că cipul E2K, rulează cu o viteză de câteva ori mai mare decât un cip aflat în cercetare la Intel și ar putea ajunge pe liniile de producție în trei ani dacă finanțarea proiectului va fi asigurată. Dar echipa rusească, Elbrus International, nu a construit încă un cip E2K, existând doar proiectele și simulările. Echipa a mai declarat că cipurile vor fi rapide și ieftine, chiar dacă construcția unei unități de producție va costa peste un miliard de dolari. Fondurile necesare finalizării proiectului se ridică la cel puțin 60 de milioane de dolari.

Intel dezvoltă un nou procesor integrat

Chiar dacă compania americană crede că acest an a fost cel mai bun pentru industria calculatoarelor, de la mijlocul anilor '90, principala preocupare a companiei Intel din acest moment este dezvoltarea unui nou procesor integrat numit "Timna". Un procesor de acest tip va integra un miez CPU cu dispozitive de control al memoriei și graficii, permițându-i să fie folosit în calculatoarele ieftine și alte echipamente cu preț scăzut. Apoi trecerea la tehnologia de 0,18 microni, care duce la diminuarea dimensiunilor procesorului, îi va permite companiei să-și reducă semnificativ costurile de producție prin mărirea cu 30% a numărului de cipuri produse cu aceeași cantitate de silicon.

Super G la 1GHz

Industria PC-urilor a ajuns la un moment istoric și anume atingerea vitezei de 1 GHz pentru viteza unui procesor. Primul PC având un procesor la 1000 MHz a devenit realitate, noul procesor fiind obținut prin overclocking de către firma KryoTech. Procesorul denumit Super G este de fapt un AMD Athlon 700 așezat într-o carcasă răcită special, lucru care permite efectuarea overclocking-ului fără pericolul distrugerii procesorului. Prețul unui pachet Super G (procesor, placă de bază și carcasă) este de 2450$ (în Statele Unite ale Americii).

Viitorul Pentium III

Surse interne ale companiei Intel relevă faptul că viitorul procesor Pentium III cu frecvența de 750 MHz va fi lansat pe piață pe data de 10 Ianuarie 2000. Din aceleași surse reiese faptul că versiunea la 800 MHz a aceluiași procesor va fi lansată pe piață la sfârșitul primului trimestru al anului viitor.

Coppermine cu probleme

Pe data de 2 decembrie compania Dell a hotărât să își oprească livrările de calculatoare desktop Optiplex GX110 datorită unui bug descoperit în ultimele cipuri Pentium III de la Intel. Calculatoarele care folosesc aceste cipuri nu pornesc de fiecare dată când butonul de pornire este apăsat.
Dell a declareat că toate livrările Optiplex 110 au fost oprite și vor rămâne așa până la determinarea cauzei acestei probleme. Între timp, Intel a declarat că problema a fost descoperită doar la câteva procesoare din familia Pentium III, iar utilizatorii nu au raportat încă nici o problemă.
Problema a apărut într-un moment delicat pentru Intel, care a fost din nou depășită de AMD (AMD Athlon 750 este net superior față de Intel Pentium III 733) și care se confrunta cu imposibilitatea satisfacerii tuturor comenzilor, datorită numărului mic de procesoare produse pe baza tehnologiei Coppermine. Totul a durat o singură zi, Dell reluând livrarea acestui tip de calculatoare pe data de 3 decembrie.
Problema amintită a fost descoperită în urma unor teste efectuate într-un laborator.

Noile G4 de la Apple

Apple Computers a îmbunătățit linia Power Mac G4 prin înlocuirea plăcii de bază de pe noile modele G4 de 350 MHz cu un design mult mai avansat. Compania a adăugat și carduri grafice mai rapide la toate modelele G4 și a lansat un nou monitor plat, cu cristale lichide de 15". Noua versiune 350 MHz G4 pune capăt efectiv vieții scurte a modelului de placă de bază cunoscut sub numele de *Yikes*. Noul design al plăcii de bază, numit Sawtooth, va fi folosit de acum înainte în mod exclusiv în toate calculatoarele produse de Apple. Potrivit oficialilor Apple, G4 la 350 MHz este superior unui Pentium III la 700 MHz

IV. REVISTA presei

PC Report

În momentul în care citiți aceste rânduri numărul 87 al revistei PCReport va fi deja pe piață. Și de această dată redactorii vă propun o mulțime de articole interesanteprin intermediul cărora veți putea realiza un prim contact cu sistemul de baze Oracle văzut prin ochii unor utilizatori FoxPro, puteți afla ce se poate și ce nu se poate calcula din punct de vedere al teorie complexității sau puteți studia prezentarea primului produs Microsoft tradus integral în limba română.

Rubrica Actual vă așteaptă cu știri de ultimă oră din domeniul IT&C; veți putea afla care a fost câștigătorul premiului Best of Comdex, care este cel mai rapid procesor produs de Intel, cine va reprezenta România în finala mondială a concursului ACM, detalii despre Microsoft DevDays '99 și HP Industry Day, precum și alte știri de ultimă oră. Utilizatorii de aplicații CAD nu sunt nici ei uitați, redactorii PCReport prezentând soluțiile Autodesk Design 2000. Nu lipsește nici limbajul de programare Java, în acest număr fiind prezentată tehnologia Java Server Pages.

În cadrul rubricii PC Concrete sunt prezentate unele probleme cu care s-au confruntat unii cititori, precum și soluții (prezentate tot de cititori) la diferite probleme semnalate în numerele anterioare ale revistei.

PC Magazine România

Revista PC Magazine România a pășit în cel de-al doilea an al existenței sale. În acest număr aniversar veți găsi un clasament al celor mai influente companii din domeniul IT&C, veți afla care sunt tendințele generale în acest domeniu și veți putea alege o modalitate de a vă petrece timpul liber îmbogățindu-vă cunoștințele de IT.

S-ar părea că în ultima perioadă Internet-ul schimbă totul, inclusiv lista anuală a celor mai influente companii IT&C. Dacă doriți să aflați cine a urcat, cine a coborât sau care sunt noii sosiți în top citiți articolul lui Jack Kirchner care vă propune, alături de acest clasament și o imagine mai detaliată asupra acestei industrii.

Evoluția rapidă a tehnologiilor oferă sistemelor portabile viteză de calcul sporită. Mircea Sabău vă prezintă promisiunile făcute de furnizorii de calculatoare portabile de pe piața românească cu privire la îmbunătățirea raportului preț/performanță.

Frank J. Derfler Jr. afirmă în articolul *Trăiască noul switch* că hub-ul a murit din cauza faptului că prețurile switch-urilor au scăzut considerabil. Vă sunt prezentate șase switch-uri cu administrare și cinci fără a putea decide care dintre ele ar putea înlocui hub-urile dumneavoastră.

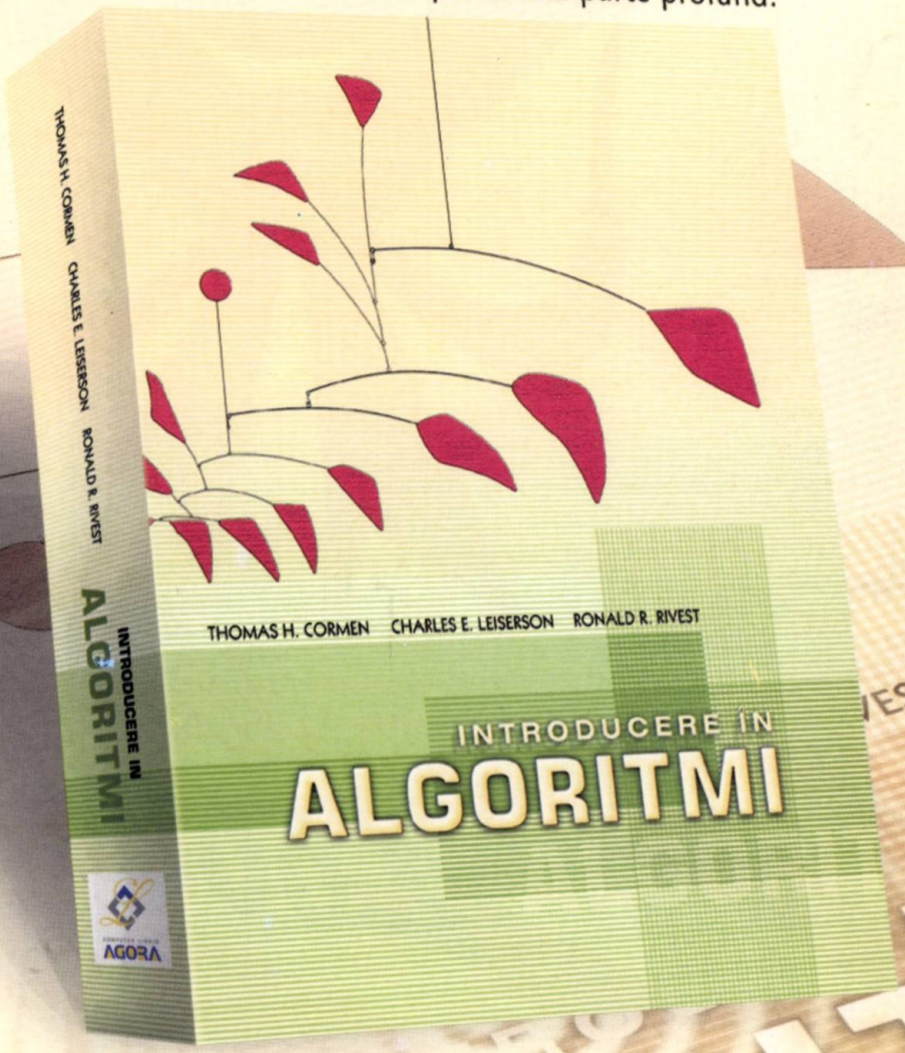
În cadrul rubricii *Prima impresie* sunt prezentate ultimele produse hard și soft cum ar fi: Hitachi DVD-ROM GD-2500BX, boxe SoundWorks, Cambridge, Altec Lansing și Sonigistix, Compaq Aero 1530, HP CD-Writer Music, Jornada 430se, Easy CD Creator 4 Deluxe și Mathematica 4.0.

CD-ul care însoțește revista este dedicat, în mare parte, jocurilor menite să mai descrețească frunțile cititorilor.

Secțiunea *Software românesc* prezintă un program pentru învățarea regulilor de circulație, unul de contabilitate, unul pentru calculul pierderilor de căldură în clădiri, precum și un dicționar român-englez. La secțiunea *Aplicații Demo* apare noua versiune a editorului de pagini Web, HoTMetal Pro 6, precum și programul pentru realizarea mixajelor audio Mixman Studio. Nu lipsesc nici aplicațiile Shareware, de această dată apărând un utilitar pentru crearea și redarea fișierelor MP3, unul pentru administrarea acestora, un program pentru administrarea spațiului pe disc, un editor de clipuri QuickTime și un editor de GIF-uri animate. Pentru utilizatorii sistemului de operare Linux există câteva biblioteci de funcții și un client ICQ pentru mesagerie rapidă. Pentru navigatorii pe Web este inclusă versiunea în limba română a programului de navigare Opera. Tot pe acest CD veți găsi ultimul număr al revistei agora ON line, Muzeul virtual, precum și un concurs destinat amatorilor de muzică.

Este în curs de apariție și în limba română celebra carte de ALGORITMI scrisă de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, publicată de MIT Press, SUA. Este o carte care de la prima ei apariție a fost retipărită, numai în Statele Unite, de 21 ori.

Această carte oferă o introducere exhaustivă a studiului modern al algoritmilor. Analiza și proiectarea algoritmilor tratați este accesibilă și asigură condițiile unui studiu, pe de o parte fundamental, pe de altă parte profund.



COMPUTER LIBRIS AGORA

Pentru comenzi sau informații suplimentare puteți suna la tel. 064-192422

GAZETA DE INFORMATICĂ
info
Teoria
La mulți ani!
2000
Gazeta de Informatică